USENIX

PROCEEDINGS of the Large Installation System Administrators Workshop

Philadelphia, Pennsylvania

April 9-10, 1987

USENIX

PROCEEDINGS of the Large Installation System Administrators Workshop

Philadelphia, Pennsylvania

April 9-10, 1987

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 U.S.A.
510/528-8649

ACKNOWLEDGEMENTS

Sponsored by:	USENIX Association 2560 Ninth Street, Suite 215 Berkeley, CA 94710 510/528-8649
Program Co-Chairpersons:	Rob Kolstad, Ph.D. CONVEX Computer Corp. {allegra,usenix,uiucdcs}!convex!kolstad Alix Vasilatos MIT Project Athena alix@athena.mit.edu
Proceedings Editors:	Rob Kolstad, Ph.D. CONVEX Computer Corp. Alix Vasilatos MIT Project Athena Tony Sanders CONVEX Computer Corp. {allegra,usenix,uiucdcs}!convex!sanders
USENIX Meeting Planner:	Judith F. DesHarnais USENIX Conference Office P. O. Box 385 Sunset Beach, CA 90742 {ucbvax,decvax}!usenix!judy
Proceedings Reproduction:	CONVEX Computer Corporation

Table of Contents

Thursday, April 9, 1987

Software Distribution and Synchronization

Mail and News

I/O Management

Closing Discussion

Administration of a Unix Machine Network

Denis Joiret, System and Network Engineer
seismo!mcvax!inria!inria.inria.fr!joiret
INRIA, France

INRIA (National Institute for Research in Computer Science and Automation) is not only concerned with both fundamental and applied research, but also with the organization of international scientific exchanges and with the furthering of knowledge-transfer and the validization of research and standardization. INRIA employs some 700 researchers (including undergraduate students) spread over three centers. I work at the center at Rocquencourt, near Paris.

Originally, this center used large mainframes such as Multics. Subsequently, several supermini-computers (Vax, Perkin Elmer) were added. They were linked through a low-speed lines network, then by a small-scale Ethernet. Most recently, with the arrival of workstations (Sun, Apollo, SM90, SPS/7), the computer facilities have become distributed. They are linked through an Ethernet network for high-speed links and a private X.25 network for terminal concentration and external access through Transpac. An Apollo Domain network and a Hyperchannel network are connected by gateways to the Ethernet network. The network involves 130 Unix-based computers (BSD 4.2 and System V) and uses DARPA protocols.

As the environment is no longer centralized, most of the operating tasks that were previously carried out on Multics by a team of operators and system administrators (registering of new users, saves, new releases, e.g.) are now performed (with varying degrees of success) by the researchers themselves on their own workstations. Our short-term aim is to provide the researchers with a work environment which includes the installation of tools on their workstations to save them the trouble of doing this work themselves. In order to do this, we are currently working on the following points:

- sharing of various hardware facilities (fast printers, tape drives, etc) and software facilities (libraries, sources, new releases),
- automatic or demand incremental dumps of workstation files (made on numerical optical disks),
- interconnecton and management of the different networks used by INRIA's sites.

To provide these services, we considered it vitally important to be able to identify with certainty each individual user. We choose to give each user a personal UID, which he would use on all the machines for which he has an account. To bring this about, we defined a two-level administration: one global and the other local. The former controls the UIDs and the names for the electronic mail; the latter controls everything else (login names, GID) locally on each machine. Tools are being developed to allow local administrators to register new users on their computers. by consulting a data base located on a server computer, in order to ensure uniqueness of the UIDs. A recently purchased Pyramid computer will provide these services. NFS will support disk-sharing.

Concerning dumps, the reading/writing on optical disks is already operational. We are using Dorofile software developed by the French company Dorotech. A backup server model is being developed on a Sun computer. When the tests have been completed, an optical juke-box will be incorporated in the system. At present, we still haven't chosen the criteria to select which files have to be saved or not (is it necessary to save Lisp core images or not, for example).

The connections between two of INRIA's three centers is already operational, using SPS-7 gateways. The chosen method consists of having IP datagrams transferred inside X.25 packets, using Transpac, in agreement with RFC 877 using leased lines if necessary for speed.

A Case Study of Network Management

M. K. Fenlon
ihnp4!ihnp3!mim
MKF IH 45262 4147 1B-224

Introduction

When a group of machines needs to operate as an integrated unit, network management becomes increasingly important. Operating in a distributed environment presents problems different from a single system or a group of independent systems. These include: security, updating and control of software, heterogeneity, link management and problem determination or isolation. The Network and Computer Technologies department of AT&T Bell Laboratories is using a trial to learn more about the operation and administration of computing in a distributed environment.

Description of Network

The AT&T 3B2 computer, a small multi-user system, was chosen. There are two 3B2 model 400 machines configured with input/output ports to handle console logging for the 27 3B2 model 310 machines, which users access. Each model 310 has a 72 meg hard disk. The 3BNET is the physical medium. Users have terminal-to-host access via the Datakit VCS network.

Remote File Sharing (RFS) is part of the UNIX System V Release 3 software available on the 3B2 machines. The RFS access allows file systems on other machines to be available to the user on the local machine as if the file systems were mounted locally. The RFS interface provides transparent access to remote system files. The file system of machines on the network can be mounted and will be treated at the user level just as any other mounted file. A server-only implementation was developed for the mainframes in the trial; the mainframe shares its files with the 3B2; but cannot itself access the 3B2 file system.

A remote execution capability was developed to allow the the user to run commands on other processors. Rather than login to multiple machines, the user has only one environment consisting of file systems of many machines.

Besides interactive network software, batch service is also available to access machines that do not have RFS software. To move data outside of the cluster, batch networking is used. The user is unaware of any media dependencies. A request may use *uucp* with Datakit VCS as the medium or RFS to a large processor machine with a high speed interface that sends the job through the established batch network. The user interface is constant but the network medium varies depending on the machine receiving the file.

Issues for Distributed Computing

Some issues involved in distributed computing are similar to those found in uncoupled environments. Security and problem determination or isolation are concerns for any system administrator. Distributed computing adds some new dimensions to these concerns. New issues also arise only in a distributed environment. Heterogeneity, maintaining and updating software, and link management are new problems that face a network administrator or manager.

Logical access, physical access and administration must be handled with security of the environment in mind. For logical access to a remote file, RFS uses the same means for limiting access as is used locally. Users can only read, write, or execute files that have the appropriate permission settings. Physical access is an open concern. With the current trial user community, the users need the processing power available on all the machines; they have no need for the machines to be co-located. All machines are secured physically; users have terminal to host access via the Datakit VCS network.

Currently, methods other than CSMA/CD are being investigated for networking in a distributed environment. The requirements driving this investigation are to secure the medium from tapping and to provide for better medium network management. Besides physical and logical access control, there is central administration of all machines in the environment. Each node in the network sends its console log to a central administration node. Users do not have to do any administration of their workstations and do not have the super-user password.

Maintaining common software across all machines in the network is important for support of users. The kernel software and tools normally found in the UNIX system need to be the same on all machines. A mechanism exists to audit the software packages in the environment. To be audited, the package must be entered in the package database. The entry will include a listing of all the pieces of the package and checksums for determining if some change has been made. Besides this standard environment, some packages are available on a request basis. Efforts are underway to fully automate the processing of requests for software installation.

Heterogeneity presents a challenge to providing a unified service environment to the users. If a distributed environment is to work, currently separate functions have to be brought under one service effort. There are several aspects of heterogeneity in the environment. First, different host types exist: large processor UNIX operating system machines, the 3B2 model 310, and a 3B2 model 400. Although all machines are running the UNIX operating system, the version for the large processor machines is different. Second, the large processor UNIX operating system machines are administrated separately from the 3B2 machines. Also these machines have users that are not part of the 3B2 environment. These users may have different objectives and priorities for service. Third, although interactive networking is a key component, batch networks also are used that have different underlying media. It is important to think of the collection of machines and network media as a single environment because the effective functioning of the environment depends on its component parts and the integration of all components.

Users will become increasingly dependent on the network and other machines within the network. With resources distributed throughout the network, a global approach for handling links between nodes is required. Frequently, determining the source of a link failure is more of an art than a science. Any failure of a link could be either physical or logical in origin.

In the trial environment, physical connectivity is through 3BNET. The physical connectivity has been more stable than the logical connectivity. Any physical link problems will of course cause a logical link failure. Therefore, to determine the source of any link failure, problem isolation skills are required. When focusing on the possibility of a physical failure, the medium monitor is useful. It can display statistics for many nodes in the network concurrently and in real time.

The logical level of connectivity refers to the software used to allow nodes to communicate. For interactive networking using RFS, the logical link between nodes is maintained continually using network protocols. This means even when users are not moving data, there is a base line of network activity. Having N by N connectivity (each machine able to connect to every other machine) is not feasible because too many resources are used in maintaining the connectivity. Besides, users do not need continual connectivity. In the trial, the system administrator controls the mounting of a standard Users request other sets of resources. At any one time, the typical machine has 5 or 6 links to the standard resource set and 1 or 2 resources requested specifically for the machine by the users.

An overall network management approach is required for determining the source of a problem. The functioning of the total environment is the concern. In a distributed environment, problem determination is challenging. A problem may not be easily isolated to a single machine. A change in the kernel or tuning parameter of a node may have an effect on the network performance. A growing workload may influence how well a network functions. A hardware problem may cause spurious noise on the network. Although one person cannot handle the details of each machine as well as the network as a whole, a single point of contact for the network environment is desirable to interface with the system administrators and operators to solve problems.

Conclusion

A network of machines must be managed for effective use of resources and to achieve user objectives. Rather than having a collection of machines, the manager has an environment whose operation needs to be maintained. Management of a network needs to be concerned with global issues such as software distribution, security, heterogeneity and link management. The network manager needs to facilitate the solution of problems that influence the usefulness of the overall network environment.

Excelan Administration

Thorn Smith
mtxinu!excelan!thorn
(408) 434-7484

My employer, Excelan, manufactures a wide range of computer to computer ethernet hardware and software ethernet products: mini's to micro's. Our new LAN hardware and software products are operationally tested and developed on our own running machines, making system administration more difficult than normal.

Our network comprises a VAX 780, 2 VAX 750's, and a dozen 8-32 USER ISI boxes all running BSD 4.2 UNIX. We also have 80 ATPC's and assorted test Sun's and Apollo's, Micro-vaxes, dual controller network running TCP.

The programs /etc/dump and /etc/restore have been customized to work across the network via pipes in the background nightly to be "dump-taped" later during the day. A dedicated machine called **DUMP** running single-user does nothing but poll the other machines and run dump-to-tape on the console.

Another dedicated machine called "Excelan" is our communications machine. This machine processes Usenet, UUCP, notes, and a custom BBS shell for customers. Our mail configuration file is heavily customized to automatically reformat UUCP mail addresses to be routed only through this central machine.

The network administrator maintains a database of employee information that is sufficient to derive /etc/passwd files, /etc/hosts files, mail aliases, phonebooks, mail-stop lists, organizational charts, and other miscellaneous documents using a top-to-bottom processing architecture. Using a master database to generate system control files and organizational and salary/stock printouts presents both problems with security R/W access levels within the database, as well as the converse burden of responsibility upon the network administrator (who often finds himself running more than just the computers!).

Using the top-to-bottom approach from a database means that all changes must be made to the master database and a regeneration of the files must be ran. Any software that previously modified the generated files, such as the "gecos", commands did with /etc/passwd; can no longer be used since no mechanism exists to "decompile" the file changes back into the database. However, generating from a non-overlapping and central source prevents problems such as contention and trans-network file locking from occurring.

Another problem with programatically generated control files is that most of them had to be split into two halves. In the case of /usr/lib/aliases, the "top half" (boiler plate) is still manually maintained, whereas the "bottom" half is computer generated and installed by a file merging script. In the case of **/etc/passwd**, the file had to be "sideways" split in such a way that the passwd field is manually maintained (via **/bin/passwd**), and the other fields are generated and installed programatically. The passwd file has to be processed in this manner because the encryption cannot be generated by the database.

Balancing Security and Convenience

Von Jones
David Schrodel
ihnp4!convex!{vjones,schrodel}
Convex Computer Corp
Dallas, Tx

Convex Computer Corporation is a mini supercomputer manufacturing company. The user base at Convex ranges from software design engineers to computer illiterates who use the computer only for electronic mail. We support hardware and software design and testing, customer benchmarking, new user training, and field engineer training machines. Security on these UNIX systems has been has been challenging to maintain. Our biggest obstacle is that different groups want different levels of security. The hardware and software design engineers want reliable computers and a great deal of freedom to move around their machine but require isolation from the rest of the world. The hardware and software test group also needs freedom of movement and isolation but does not have and does not promote reliability.

The users who perform customer benchmarking, training, and accounting want isolation from other users on the same machine and impeccable reliability. It is an ongoing struggle to balance the amount of security desired and the amount of user inconvenience which this level of security will cause.

When setting up a new machine the following questions need to be asked:

- How secure must we be from the outside?
- How secure must our machine be from users on other machines?
- How secure must we be against users on the same machine?
- How secure must we be against interruption of computer services?
- How secure must we be with respect to user error?
- How much inconvenience are the users of this machine willing to endure to support this level of security?

Generally, a secure environment is one that will protect data from being violated by outsiders or insiders without proper permissions, will provide services without interruption, and will protect users from user error (e.g., some novice user might unknowingly type **rm -r /***). Some of the common problem areas are file accessibility, machine accessibility, invalid user access from internal sources, invalid root access, invalid access from persons outside the scope of the company, and machine reliability.

File accessibility must be controlled by the operating system. Within UNIX we have found some methods for controlling file access which are very reasonable to set up and maintain. File security is dealt with primarily by means of the UNIX protection mode bits. Important system files are always protected. If for no other reason, this guards against many catastrophes caused by user error. Users such as third party software development, who desire protection from other users on their machine, can be placed in groups according to the product(s) on which they work. Permissions can be used to regulate access. On their machines it is necessary to restrict root permissions to a small handfull of users. If there is only one user in an account, we encourage users to deny permissions to other groups. This is generally sufficient for their needs. There are, however, instances where permissions cannot guarantee security. Managers and members of the personnel department require protection from all other employees (even employees with root access). For these users, file encryption becomes necessary. At Convex we routinely remind managers and personnel employees of our encryption routines. This allows file security with a small amount of effort. We have yet to have any problems with people breaking encryption codes to find out what is in a file.

Restricting machine access is possibly the most usability-restrictive of all of the security measures which we have implemented. This machine security relates to machine networking. Depending on how the restrictions are set up in the files like **hosts.equiv** and **~/.rhosts** user accessibility can be affected greatly. When setting up the machine for network use, we generally look at who will be most likely to be use a machine.

For example, all of our hardware and software development machine are "trusted" hosts and allow *rsh*'s, *rlogin's* and, *rcp*'s without passwords. The customer benchmark machines, on the other hand, have users from many different companies working on them. To protect the development machines from invalid user access from the "untrusted" hosts, we do not include those machines in the hosts.equiv file. This of course decreases the usability of the marketing machine, but all affected users must agree that these are reasonable steps to insure the integrity of the other machines. The one function that most people missed was the ability to use the printers on the development computers, since the line printer daemon on BSD UNIX checks the hosts.equiv for hosts that it will accept jobs from. To overcome this restriction, we changed the line printer daemon to accept a flag telling it what file to take hostnames from for accepting jobs. Such changes as this are easy to implement and yet can make a network security restriction much easier to live with.

Invalid user access from internal sources has not been a problem, but we have nevertheless installed several software changes to decrease the likelihood of invalid user access. For internal users the main security problem is not one of someone sitting down and guessing another user's password but rather one of a user walking off and leaving their terminal logged in or doing such things as programming his terminal function keys to enter his user name and password; it is a problem of physical security. To cut down on the window of time a potential break-in can take place, we installed autologout features. Admittedly, this is not fool proof, but generally the problems are going to be at night when most people are not at work. This also has a side benefit of freeing up ports into the computer if some sort of front end terminal switch is used. We also encourage users to take responsibility for the physical security of their own station.

Root permissions pose a large security risk. Since root can access any file, keeping root out of an area of a filesystem is impossible. We have addressed the root security problem by extending our accounting capabilities. Allowing people to log in as root is like sharing user accounts – it becomes impossible to single out an offending party. We must, however, give quite a few users root privileges. To try to control the problem as the company has grown, we use a setuid root program which when executed by "privileged users" will give them a root shell. This allows us to at least know who was becoming root and usually track down the user who did something if the need arises.

Protection from persons outside the scope of the company takes two forms: physical protection and software protection. Physical protection measures include alarms, guards, and other means of denying entrance to facilities which contain computers, terminals, computer wiring, and data storage. The harder part of denying access to members of the outside world is the security of dial-in lines provided by software. To keep someone from breaking a password by trying many different passwords, we installed a dial-in password so that on any modem line, a password would have to be entered in addition to the normal user name and password. Upon entering an incorrect dial-in password, the line will drop and force a person to call in again. Care is taken to make sure that the dial-in password is not in the dictionary. Usually the system breaker couldn't break in because of the time necessary to redial while trying passwords.

The machine reliability issue is fairly easily dealt with. We have addressed this issue by grouping together on the same machine those users to whom reliability is not important and those users whose jobs tend to cause unreliability. Usually this means hardware and software testing is restricted to a specific group of machines. We have established a sort of spectrum of reliability across our machines. Testing is done on the most unreliable machines; processes such as accounting and development are done on our most reliable machines.

All of these security measures require time to implement. Some of them require a great deal of time. Often the trade-off between time and security does not merit putting the time in to make a system fully secure. The bottom line to all of this is that when you set up security policies for your systems, you must decide how much inconvenience your security is worth to you. If security is a number one concern, then it would be worth your time to eliminate hosts from hosts.equiv, remove root access from all but the most trusted people, and establish robust accounting procedures. In many cases, however, time and the inherent insecurity of UNIX can force us to reduce our ambitions to keeping the outside world out and trusting our employees. In most instances all of the security in the world will not keep out a determined individuals, and too much security can cost many man hours in inconvenience and will only give you a false sense of security. The key is to determine the time-security balance necessary for your application, and to work to implement that balance.

Automated Dumping at Project Athena

Alix Vasilatos
alix@athena.mit.edu
Systems Support
Project Athena, MIT

Project Athena is an experiment in the uses of computers in education, financed by DEC and IBM.

We are currently running a Proteon fiber spine with local sub-ethernets; some subnets gateway to the spine via 9600 baud links. The machines (VAX 11/750s, VSIIs, and IBM RT/PCs) number around 400 with an expected growth to 1500 over the next year. In Systems Support, we have the one lead programmer (me), four full-time systems programmers, two operators, and 11 part-time student programmers.

System maintenance in a mixed, large environment like this can be a nightmare. Our backup scheme has been very effective. Without showing you the complex innards, I think the ideas are right and we have had amazing success restoring blown disks. (In the past year we've lost 30-40 RA81s and RA80s so badly that they were unreadable. We've not lost more than 8 hours of data that I can recall.)

The large dump scheme is just for the 750s. We have grouped the 750's into groups of 5-8 systems (each group in its own machine rooms). We call these groups clusters (not TRUE clusters, but that's what we call them). Each cluster is on its own subnet and communicates through a MicroVax II gateway to the spine. There are 8 clusters of timesharing systems.

Each cluster has just one "server" system; all the other computers are 1 RA81 "user" disk, where folks' stuff lives. The RA80 contains the system software, such as it is. The server has 3 RA81s (at 400Mb each) and a TU78 tape drive.

Every night, via crontab at 3 am, the server system runs a program called **backup** which reads a list of hosts it has to back up. One host is always selected for either a level 0 full dump, or a (relatively large) level 5 incremental dump. That host is put on hold for a moment, while all other hosts "rsh dump" themselves at level 9 into files kept on the server's ra1, the second RA81. These files remain on disk until they are overwritten by the next level 9 the next night.

That leaves the candidate for the big dump. This dump is performed at the appropriate level (0 or 5) and is dumped to a file on the server's ra2 (a C partition; the whole disk). The next day an operator visits all 7 clusters and pulls this dump off onto tape, using a program called "dumptape" (which, incidentally, writes the tapes with dd using a 63K blocksize).

At this point, we have big dumps on tape with labels generated by dumptape and small current incremental dumps on the server's disk. The next night, the whole thing starts over again, where the next system to be honored with a large dump is identified by the backup program and moved up in the queue.

If we lose a system disk (ra0) on a 750, we copy over its neighbor's RA80 and fill in the site-specific blanks from our site-specific file maintenance system. (It's really quite simple; we keep correct copies of files like /usr/lib/aliases and /usr/lib/crontab in a file system on a single central 750, which we back up.) A full restore of an RA80 system disk from scratch takes about 2 hours.

The easiest of all is if we lose one of the 340 VaxStation II's or IBM RT/PCs floating around in public areas. We just reload the entire system from installation floppies; since the machines don't have enough space for people to keep their stuff on them anyway, we don't worry about backing up user files. It takes about a half hour.

There are some problems, though. One problem is bad tapes: with a blocksize of 63K, you lose a LOT of what you need when just one weensy section of tape is messed up. If operator who is supposed to pull these things off onto tape doesn't, the system continues to run and disks fill. UNIX then gets upset and prints reams of "file system overflow" messages.

System Backup in a Distributed Responsibility Environment

Carlton B. Hommel
masscomp!ogre!carlton
Software Release Engineer
MASSCOMP - Westford, Ma

MASSCOMP designs, manufactures, sells, and services microsupercomputers used primarily in real-time data acquisition and technical computing applications worldwide. Our internal engineering network consists of over 150 MASSCOMP computers running on one TCP/IP network. Most are single-engineer workstations but there are about 20 high-end machines residing in a raised-floor computer room that must be backed up on a regular basis. There is one software aide who dumps systems part-time.

Everyone is familiar with the difficulty of getting individual engineers to do daily backups. At MASSCOMP, this is aggravated most workstations having only 1/4" cartridge tapes. Waiting for a dump cartridge to rewind is beyond the pale for most programmers - until their first crash. We needed a *dump(8)* frontend that was robust enough to deal with a multi-Eagle archive machine, inobtrusive enough for a hardware engineer, and smart enough to keep an aide from doing an `rm -rf /`.

Backups are divided into three classes: monthly, weekly, and daily. The "Tower of Hanoi" method, mentioned in the man page, is not used. Instead, all monthly dumps are at level 0 - the whole filesystem. Weekly dumps are done from levels 1-5, and daily dumps are done on levels 6-9. The *date(1)* command is used to determine which level.

The *cron* program does the nightly dump at midnight without operator intervention. The advantage to this method is that you can keep the same tape inserted in the drive all week, and still have nightly protection. There are several disadvantages:

- A exremely volatile filesystem will overflow the tape by Friday morning.
- An interrrupted dump will destroy the previous day's backup.
- Someone may forget to replace the tape after using the drive.

The script checks whether a tape is in place before running the *dump(8)* command.

Weekly backups are done Friday morning (during a weekly status meeting) with the backups -W command, and the weekly dump tape.

Thus, to restore filesystem on the Friday the 31st of the month would require just three *restore(8)* commands - one level 0 Monthly, one level 1 Weekly, and one level 5 Daily. This is always predictable, where a Tower of Hanoi approach would vary as of the phase of the moon.

The script also makes allowances for dumping multiple filesystems on one tape. For example, our OS archive machine has a nightly dump for both the root and /usr/src filesystems. This is done by using the "no rewind" node of the tape drive. (/dev/nrmt0, rather than /dev/rmt0.) In this case, careful use of the "mt fsw" command will position the tape before the appropriate dump image.

The script also allows the nightly dumps to be done onto a different device than the others. For example, the Release Engineering machine does its nightly dumps to 1/4" cartridge, while the operator-intervention dumps are onto the 1/2" tape drive.

The script is still being worked on, and I am overcoming inertia among engineers who have the 8:45AM full dumps hardwired into their brains, along with their 8:46AM coffee break. However, the convience of automatic nightly backups and a software aide to run this program are helping.

Restoring from Multiple-Tape Dumps

Harris Jaffee
ihnp4!princeton!idacrd!hj
Institute for Defense Analyses,
Communications Research Division
Princeton, New Jersey 08540

The computing facility at IDA/CRD includes a network consisting of a VAX 780 and several workstations. An operations staff supports the facility around the clock. The VAX is used mainly for program development and document preparation. The number of users is fairly small. This select group of users is somewhat spoiled regarding processor performance, the capabilities of system utilities, and disk usage.

The VAX 780 runs BSD UNIX, and has rm05 (256 Mbyte) disks. We use a non-standard disk partition which allows one of the file systems to be about 212 Mbytes. A full *dump(8)* of this file system may require up to five 2400 foot 1600 bpi tapes. The *restore(8)* program is not very convenient for finding 1 file in a 5-tape dump. Neither the users nor operators are impressed very much with a program that says: "If you don't know which tape to use, start with the last one ..." With smaller partitions which can be dumped to one tape, the problem never arises. Actually the problem belongs less to *restore* than to *dump*, which we have improved.

We made a small change to dump to write (on a file) the inode number of the first file on each tape. (This is at the end of otape(), on dumptape.c, where it writes this information on standard error.) We accumulate this information in a permanent database, along with the date and the name of the file system. We keep on-line for a while the output of "restore t" for each dump, which lists the files dumped, together with their inode numbers. Since files are dumped in inode order, it is trivial from the inode number to select the appropriate tape.

We have taken one step further. Frequently, a user does not know the exact name of the file(s) he wants restored. We have shell scripts which wander around in the "restore t" outputs offering the user a list of files whose names start with the user's name and contain a specified string. These scripts supply a tape number, given a file name or inode number, and run restore automatically. When restore asks you to mount another tape, there is no guesswork nor waste of time.

An alternate approach to this problem might be to have dump write its "directory" at the end of the last tape, in addition to the beginning of the first tape. At that time it would also record these "first file" inode numbers, which of course it didn't know when it wrote the first tape. Then you could start restore on the last tape, and change to an earlier tape if necessary. The point is that restore would be able to tell you which tape to mount.

Backup and Restore for UNIX systems

Mark Poepping
sun!sei.cmu.edu!poepping
Senior Software Engineer
Software Engineering Computing Environment
Software Engineering Institute
Carnegie-Mellon, Pittsburgh, Pa.

The Facility

The Software Engineering Institute is an FFRDC (Federally Funded Research and Development Center) sponsored by the Department of Defense under contract at Carnegie-Mellon for the purposes of advancing the state of the practice in software engineering principally among the defense department contractors.

The SEI computing facility consists of ~100 MV-II's, a dozen Sun-3's, five Apollos, a bunch of Macintoshes, several PC's, and assorted other hardware. Maintenance of the Vaxes and Suns is the primary responsibility of the facilities group. We are currently running a 10Mb Ethernet at our main facility with two satellite 10Mb net connect by routers through 4800 baud modems. The first satellite is several blocks away, the other is in Massachusetts.

We have sixteen people on the facilities staff: a manager; a secretary; three senior programmers, one each to major environments (Ultrix, Sun3.0, VMS); two hardware/wiring specialists, an operator/technician, two student assistants, and six software engineers devoted to various development, maintenance, consulting, and documentation tasks.

The Problem & Complication

How do we handle backup and restore for the UNIX systems? With the recent availability of inexpensive disk subsystems for the workstations, our total disk capacity has risen to over 20 gigabytes, with 15 GB eligible for backup. Typical workstations have about 200MB of local storage, with some running up around 1GB. We must therefore treat the workstations as real machines and include them in all backup/restore operations.

The Solution

We are currently performing weekly full backup and nightly incrementals using about one-half of one person (operator / technician). Here's a short description of the system in place:

Full Backup

The backup procedure is a two-step process. For full saves, each workstation has been assigned a unique time and day of the week to perform the backup. At the specified time, the workstation copies the data to be saved across the network to the backup server. The next morning, the operator copies the data from the backup server to tape.

The full backups are removed from the backup server after they have been written to tape. These tapes are held on site through the following week in order to service restore requests. After the week has elapsed, we save the old tapes off-site. Every fourth set of tapes is permanently stored off site; the rest are returned for recycling. In this way, we have monthly checkpoints of all systems, and we will never lose more than one week of work (barring catastrophe at the off-site repository).

The backup server is a single MV-II with four Fujitsu Eagles (1.4 GB) for spooling space. It has a single 6250 bpi 9-track tape drive.

The full backups are started from each workstation by the local *cron(8)* facility. Each local logical device is dumped separately and status is logged to a file on the backup server. There is no automatic administration of free space on the server. If the server has no space to store any portion of a dump, the whole dump will abort. It is up to the operator to notice this and perform the dump the following day, before writing the dumps to tape.

Incremental Backup

Each machine also has an assigned time for incremental saves. Each night, the incremental backup server requests copies of all controlled files which have changed since the last successful incremental. "Controlled files" are those files and directories specified in the **/backup** file of each machine. The incremental backups are removed from the backup server only when space is required to hold the next days backups. Each day, the operator checks to make sure the incrementals have completed successfully, copies the last two working days of incrementals to tape, and removes some old incrementals to free space for new ones. These tapes are recycled bi-weekly and are never stored off-site. Ordinarily, we are able to hold about 10 days of incrementals on line. The on line incrementals are used to support interactive file retrieval. Any user can use the *retrieve(1 SEI)* command to inspect or restore files from the on line incrementals.

For incrementals, the backup server is a single MV-II with three Fujitsu Eagles (1 GB) for spooling space. It has a single 6250 bpi 9-track tape drive.

The incremental backup scheme relies on *sup(1 CMU-CS)* (Software Upgrade Protocol, a local tool developed in the Carnegie-Mellon Computer Science Department). In short, each workstation has a file containing the list of those directories (sub-trees) which should be examined for new files. Each night for each machine, *cron(8)* on the backup server starts the incremental backup procedure to pick up those files which have changed since the last successful incremental from that machine. As a special service, the incremental backup facility (optionally) sends mail to each user notifying them of the files which were saved from their home directory.

There is no particular reason why *dump(8)* could not be used for this purpose, there are historical reasons (no longer convincing) why we used *sup* in the implementation.

The Evaluation

Both full backups and incrementals work extremely well, especially considering the fact that we've spent very little time implementing (didn't write much code, did think about it a lot).

There are basically three problems:

- There is a tendency to forget about the inevitable restore operations when defining hardware and staff requirements. Restore operations will usually take twice as long as the initial dump. This ties up the tape subsystem, making it unavailable for dumps. But more seriously, it occupies a lot of staff time originally devoted to the dump operations. So we are short on staff and short on hardware.
- Since we stage files on a server before spooling to tape, we've added more hardware (and more potential failures) to the path. So, if we lose our server machine, we can't do backups (or restores). The solution here is to provide more replication and a highly dynamic dumping strategy. Machines should be able to dump to any of several server machines, and the dumping should probably be controlled centrally, rather than distributed throughout everyone's crontabs (we are currently moving in this direction, while watching for other developments elsewhere).
- Since the staging area for incrementals is over an order of magnitude smaller than the total area covered by incrementals, it is very susceptible to data overflow. The primary trouble we have is due to the tools used for incrementals (we don't use dump). In particular, this local tool considers both creation and modification time when deciding to save a particular file. The "creation time" is bad since it is modified during several operations, most notably the restore operation.

Software Synchronization at the Federal Judicial Center

Julie Becker-Berlin
fjcp60:julie
Federal Judicial Center
Washington, D.C.

The Federal Judicial Center is a small agency of the judicial branch of government that primarily provides a research and development forum for the federal courts. It was decided to replace the central data processing equipment, located in Washington, D. C., with machines that would be onsite in the courts. The failure of the existing vendor to provide promised hardware and software upgrades for an anticipated increase in processing was a primary impetus for decentralization of computer resources. As part of that decision, the UNIX operating system was chosen in order to become vendor independent and take advantage of ongoing technological developments without being trapped to a particular type of architecture. Software developed under UNIX could be transferred between different systems with a modicum of changes to existing programs.

Within our current environment, the Center supports several sizes of machines, ranging from 8-user systems up to 80-user systems, with a variety of vendors being represented. Large database application development is the primary use of these machines, as well as software evaluation. Most of the development machines are on a local area network, using ETHERNET with TCP/IP and XNS. In addition to the in-house machines presently supported, technical assistance is provided remotely to the courts who act as "pilots" for the applications that are being developed.

Several problems exist in supporting the equipment both locally and remotely. Various types of problems occur in each of the sites, presenting the staff of the Center with a unique opportunity to address a wide variety of issues.

Locally, there are four major aspects that need attention: Developing common software to be ported to multiple machines; networking different machines together, each running a distinct version of UNIX and networking software; tying separate local area networks together; and managing large database file systems. The Center is developing a family of applications that will provide case management facilities and on-line docketing to the federal circuit, district, and bankruptcy courts. In each of these types of courts, similar functions can be grouped together to provide a generic series of programs common to all applications.

A primary difficulty in dealing with a multi-vendor environment has been keeping the various versions of the software in sync between one kind of machine to another. Networking the assorted machines together has helped provide a solution to keeping the versions of developed programs in step with each other. However, it has also created another series of issues regarding upgrades to systems and networking software, as well as maintaining shared peripherals on the network. Linking other networks, such as AppleTalk and ThinNet, to ETHERNET presents additional opportunities to explore the use of microcomputers within the UNIX environment. Since the case management applications are large databases, another concern has arisen regarding the management of the large file systems.

In the remote court sites, a different series of problems arise than occur locally. There are three basic areas of concern: Personnel and training issues; maintaining and troubleshooting local hardware and software failures; and management of computer and human resources. In most of the court sites, existing personnel had very minimal prior experience in the use or administration of computer systems. At most, word processing machines may have been present, with an outside vendor handling the support of such equipment. With the advent of decentralized data processing, the courts had to be assuming more of the responsibility for hardware and software operations and maintenance.

A decision was made to train remote court personnel to become system administrators instead of hiring outside computer professionals. The problems associated with a wide variety of skills and learning abilities, as well as tailoring the training to machines specificities have presented numerous challenges. Depending upon the prior experience of the court personnel, the ability to diagnose problems and maintain the equipment has varied significantly, thereby changing the level of support provided by the Center to each site. Since a number of the pilot courts did not have previous systems experience, managing the computer resources and providing time for maintenance such as backups proved to be difficult unless adjustments in personnel scheduling were made.

The solutions to these problems that arose were implemented in an evolutionary manner. As each problem arose, some kind of fix was suggested, and very often was subsequently modified and as additional experience was gained.

When the development for the large database applications began at the Center, there was no library management system for UNIX source code. More recently, a master source library was established, with one or two people designated as the "czars" for consolidating changes across the machines. The inter-machine networking facilities also helped keep versions of the software in sync. But installing new releases of the operating systems and networking software created other problems that were not always easily solvable. Unfortunately, trial and error played a major role in maintaining the network. The management of large file systems for the database applications still presents a problem because of backups, fragmentation of the file systems, and generally inefficient accessibility to files within long directories. Recent attempts to resolve these problems have focused on the evaluation of third-party vendor software for file system backups and tuning existing application software.

The system administrator training developed by the Center has evolved considerably from our first training class. The staff of the Center have been assigned to drill the new court system administrators about the daily systems tasks, hardware maintenance, and routine problem solving. A comprehensive UNIX system administration manual has been developed and distributed to court personnel. As a result of these efforts, the current administrators in the field appear to be better able to cope with the equipment that is placed in their courts than their predecessors. The Center is developing self-paced computer based training and introductory videotapes for the UNIX operating system. The courts have made adjustments by assigning personnel flexible working hours in order to support the maintenance duties for the machine.

UNIX is a trademark of AT&T Bell Laboratories ETHERNET is a trademark of Xerox Corp. AppleTalk is a trademark of Apple Corp.

Release of Replicated Software in the Vice File System

Christopher Koenigsberg
ckk#@andrew.cmu.edu
Andrew Systems Administration
Carnegie-Mellon University
Pittsburgh, Pa. 15213
(412)268-8526

Several hundred Unix workstations all use the Vice file system at Carnegie Mellon University. Each machine has its notion of a local Unix tree but most of the files are actually stored in the distributed file system. IBM RT's, Sun2's, Sun3's, and microVax 2's all share the same global namespace, which has both tremendous advantages and tremendous pitfalls for the system administrators. In this paper I will discuss the replication and release of software to campus workstations.

Each workstation's Unix kernel has been modified with the "Vice hooks", whereby any application's request to open a file is tested for the /cmu prefix. If the filename has this prefix, the request is trapped by the kernel and sent to the Venus file manager process. Venus fetches the file from the Vice file servers, stores a temporary local cached copy, and returns a pointer to this temporary local inode. The application never suspects that its file descriptor points to a temporary cached copy rather than a permanent local file.

We attempt to provide a local Unix namespace on each workstation, with the judicious use of symbolic links into the campus filesystem to provide for mail and bulletin boards, and central maintenance of system software. A small set of critical system programs are stored on the local disk, along with the symbolic links into the campus filesystem. At reboot time, a program called "package" checks the entire local disk configuration, making whatever connections and updates are necessary to match its specified configuration.

There are roughly three views of the file namespace, each one broader in scope than the previous one. The first is the view through the shell search path. The second is the view through the local workstation's Unix namespace. The third is the full campus filesystem namespace. By providing different namespace scopes, we allow users and Unix programs to name things without having to understand and specify the full campus pathnames of all files.

For example, suppose a user types "help" in a C-shell window. The user's search path includes /usr/andrew/bin, and the C-shell finds /usr/andrew/bin/help there. There is a symbolic link called /usr/andrew on the local workstation, which actually points to:
/cmu/unix/@cputype/usr/andrew.
(@cputype is an abbreviation meaning one of the following: ibm032, sun2, sun3, vax). The binary for the Help program is actually stored in the campus filesystem in this directory, but neither the user nor the C-shell have to know this much detail. The user can sit down to an IBM RT and run help in exactly the same as they run it on a Sun3, or a microVax. The symbolic link /usr/andrew points to the different cputype binary on each different workstation.

When the Help program starts up, it scans all the Unix man directories, and the Andrew help directories. It looks through /usr/man/*, /usr/andrew/help/*, and /usr/andrew/doc/*. Once again, these are symbolic links into the campus filesystem. There are both machine-specific and cpu-independent documentation files. The Unix manual pages come from the different vendors flavors of 4.2 (DEC Ultrix, Sun OS 1.1 and 3.0, IBM AIS 4.2A), so the /usr/man on a workstation is linked to /cmu/unix/@cputype/usr/man.

But the Andrew help files are written for applications which are supposed to work equally well on all cputypes, so the help files are stored in a more complicated way. /usr/andrew is a symbolic link to /cmu/unix/@cputype/usr/andrew but /cmu/unix/@cputype/usr/andrew/help is another symbolic link, to /cmu/unix/common/usr/andrew/help. So the effect as seen by the Help program on the local workstation is that /usr/andrew/help contains the same files, no matter which cputype the local machine is.

Other directories are set up in a similar fashion. /usr/local is a link to /cmu/unix/@cputype/usr/local, /usr/misc is linked to /cmu/unix/@cputype/usr/misc, /usr/edu is a link to /cmu/unix/@cputype/usr/edu, and so on. In general, there is a reasonably complete Unix tree for each cputype, starting with its root at /cmu/unix/@cputype. There is a bin there, a usr/bin, a /usr/ucb, etcetera, although some of these are actually copied onto a workstation's permanent local disk instead of just being linked there (for example, there must be an actual vmunix and venus on the local disk when a machine boots up, although they are updated periodically by the "package" program from the corresponding versions in the campus filesystem).

These directories are all on read-only, replicated Vice volumes. The replication provides for continued service even when individual fileserver machines are inaccessible. It also balances the load on the fileserver machines, because campus workstations fetch their files from several different servers instead of all depending on the same one for a crucial system file. They are cloned and replicated from Read-Write volumes, in an operation performed by a system administrator logged in to one of the file server machines. The process of releasing new software to the readonly volumes is my responsibility in the Carnegie Mellon campus system.

When a system developer compiles a new version of an Andrew system program, it is installed into an experimental subtree of the campus filesystem. We have a set of directories referred to as "andy" instead of "andrew" for this purpose. Developers' workstations are linked to the experimental volumes rather than to the released volumes. On one of their workstations, /usr/andrew points to:

/cmu/rw/unix.exp/@cputype/usr/andy

Note the three differences between this and the link on a regular campus workstation. The "rw" means that this is in the Read/Write tree rather than the replicated readonly volumes. The "unix.exp" indicates that this is an experimental place rather than a stable place. The "andy" rather than "andrew" allows us more flexibility in controlling the testing and release of software.

A regular campus workstation user can actually run the experimental version of a program by running it from /usr/andy/bin instead of /usr/andrew/bin, since /usr/andy is linked to:

/cmu/rw/unix.exp/@cputype/usr/andy

on all campus workstation. But the vast majority of our over three thousand registered users are not as curious.

When a program is deemed ready for testing and release, we copy all of its relevant files from the experimental tree into the testing tree. Certain workstations set up for testing have /usr/andrew pointing to:

/cmu/rw/unix.test/@cputype/usr/andrew

We try to provide them with an environment almost identical to the regular campus workstation, except for the applications currently under testing. Developers and testers are working together now to complete a library of brief testing scripts, one for each application which gets released regularly. Sometimes there are subtle interdependencies between applications, and these are hard to track down the first few times a release is made. For example, the Message system user interface depends on the underlying Mail system software, which also depends on something called the Guardian which runs on each local workstation, and all of these must be compatible with each other for a new release to function properly. Having a separate testing area, with dedicated test machines, allows these dependencies to be tracked down.

When testing is completed on an application, we install it into the "staging" area, a ReadWrite tree which is then cloned and replicated. For example, the Help program is copied into /cmu/rw/unix/@cputype/usr/andrew/bin Note that this is in the "rw" tree but its name is otherwise identical to the released version used by campus workstations. A program called "vol" is used on one of the file server machines to initiate the process of cloning and replicating the application across the campus fileservers.

When the replication is finished, and the volume location database has been propagated around campus several hours later, a user can run /usr/andrew/bin/help even if the primary server on which it is stored is inaccessible. The local Venus file manager has a list of several servers where the application is replicated, and it will contact the next server on the list if one fails to respond.

Maintaining a Consistent Software Environment

Helen E. Harrison
mcnc!heh
Microelectronics Center of North Carolina

ABSTRACT

Maintaining common software in a mixed vendor environment can be challenging at best. Because of the nature of our university-like environment and our need for special purpose tools for CAD design/research, we tend to acquire (or write) large quantities of local software. To complicate things further, local CAD tools are supported by groups other than system support. We also have source code for 4.3 and Convex UNIX. Altogether this means we have a particularly large amount of software to maintain. We have found that it is important to maintain consistency not only among our mixed vendor machines, but within each machine (i.e., given a binary, one can easily find its source, etc.). We have also found consistency important when several people are maintaining the same code. Over time, we have developed guidelines, written scripts, and modified programs so that we now have a working system for maintaining source code.

Background

The Microelectronics Center of North Carolina is a non-profit corporation involved in cooperative research and education that unites the resources of five participating universities, the Research Triangle Institute and a new $40 million research facility. MCNC research programs focus on integrated circuit design, simulation and testing, as well as semiconductor materials, devices, and fabrication processes. MCNC works with its Industrial Affiliates to accelerate the transfer of this basic research into commercial application. In addition, MCNC has provided a state-wide microwave network connecting the Participating Institutions, providing broadband circuits for data communication, and two-way color video channels for interactive tele-classes and conferencing.

The MCNC computer facilities include 2 VAX 8600's running 4.3 BSD UNIX*, (UNIX is a Trademark of AT&T Bell Laboratories) a VAX 11/750 running 4.3, 2 Convex C-1's (4.2 variant), a MegaOne chip tester (4.2 variant), an assortment of Microvax II's running 4.3, several Microvax GPX's running Ultrix 1.2, a couple of other miscellaneous workstations. The machines are networked together on 2 local ethernets: one for the main machines and one for the workstations. We expect the number of workstations to increase greatly in the coming years. In addition, we have an extended ethernet with approximately 80 hosts, including departments at RTI, NC State Univ., and the Triangle Universities Computation Center. (We plan to add departments at UNC-Chapel Hill, Duke, UNC-Charlotte, and NC A&T. The number of hosts could easily grow to several hundred.) MCNC has a full time System Support staff of 6 full-time programmers, 3 operators, and a technician.

Overview

Here are the guidelines that we use in maintaining our local software:

Never put any new programs in standard Berkeley places. We have created a directory structure under /usr/local for programs that are not part of the standard (Berkeley or Vendor) distributions: /usr/local/adm; for administrative programs (i.e., those a normal user would not run); /usr/local/etc for those that would otherwise go in /etc (i.e., *ubareset(8)*); /usr/local/std for programs that general users will need; /usr/local/vlsi and /usr/local/vivid for various CAD tools. Under each of these are bin, lib, and man directories. Bin is the place for programs that people call, and lib is the place for their auxiliary files and programs. We have modified man(1) to look at the user's path to find the parallel man direcories. This permits administrators of different groups of local software (and individual users) to maintain their own bin and man directories.

All local source code resides on a single machine. /usr/local/src[std,etc,adm,vlsi,vivid] hold the sources for their accompanying binary directory. Before a program can be installed it must have a man page, a makefile that will remotely distribute it to the appropriate machines, and an RCS directory. The same source should be used on both the Convexes and on the vaxes. (No local source lives on the Convexes.)

RCS any and all changes to sources. This is probably the single most important guideline. (In our case, the source code is mostly owned by bin, so it is also necessary to identify in the RCS file who has modified the program.) A revision history it crucial for keeping up with the changes that have been made to programs when new releases come along.

A few useful utilities:

- Rdist(1), with a couple of options we added, is useful in distributing and verifying programs among machines. Copious rdist scripts are run each evening verifying binaries and mailing to the appropriate alias a list of those that are out of date. There are also scripts that go through the source directories looking for RCS files which have changed and mail the last log messages to the system staff to help us keep up with what each of us is doing.
- Rinstall(1) is an extended version of install with options for remote installations. (Rinstall is basically a front end to rdist.)
- Hostclass(5) allows for machine class definitions (like mail aliases, but for machines.), for example 'convex', 'vax8600', and 'ultrix'. Rinstall will take a hostclass specification.
- Expandhosts(1) is a quick way to identify which machines are in a particular hostclass.

Our system requires nothing particularly sophisticated or complex. Many of the same guidelines apply to system source code management as well. We have followed the UNIX toolkit philosophy, putting together off-the-shelf utilities and augmenting them with shell scripts, rather than developing monolithic packages. This system works because it is simple and clear, and because we compulsively and meticulously abide by it.

System Cloning at Hp-sdd

Ken Stone
hplabs!hpsdd!ken
R & D Computer Support
Hewlett Packard
San Diego

At Hewlett Packard in San Diego, we build "graphics output devices" which are mostly plotters of one form or another. The really hard part in building plotters is the mechanics. In the last couple of years, we have started to depend heavily on CAD for mechanical engineering. Our ME-CAD setup uses HP series 300's as workstations in the lab and one machine as a file server. All of the workstations are clones off of a master machine. We also use the same workstation for EE-CAD without a fileserver (basic UNIX is the same).

At first we thought we would have to spend several hours installing UNIX on each workstation and decided that this was impossible (initially 15 seats with 10+ new seats a month for about a year. We just did not have the people to spend that much time on installation, and then still have to worry about customization of each station.

The solution has turned out to be very simple - "cloning". The key to cloning is just "cpio -pudx". We have a shell script that a tech can run that takes a raw disk and turns it into a working UNIX system that is already customized and ready to come up on our network. First, we had to deal with "customizing". Since these workstations are just high power graphics terminals for the most part, we were able to limit what we had to change on each machine to less than a dozen parameters. Each of these parameters (hostname, internet address, etc) is kept in a separate file in the directory /info and anything that needs this info was modified to get it from there (mostly scripts run at boot time). This concept also proves handy in doing system updates. Once this was in place, all we had to do was write a script that did the following:

1. Prompt tech to connect new disk drive
2. Format drive and newfs it
3. Mount it on say /ndisk
4. Clone on to new disk
   `cat filelist | cpio -pudx /ndisk`
   where filelist was created with something like
   `cd /; find . -print > filelist`
   The advantage here is that find only has to run once for many systems being cloned and you don't have to worry about cpio going recursive
5. Prompt tech for system name and do a lookup in a predetermined database to find all the necessary customization parameters.
6. Customize new system using sed

A word of warning should come here. This process makes it really easy to violate licenses. You must be sure that each system is fully licensed with all needed documentation or when your next audit comes around, the accounts get extremely mad!

One benefit of this arrangement that we did not really foresee is that in the event of a disk crash or some other such problem, our recovery time is almost nil. We can just clone up a new system. Backup strategy is also easy, just backup the fileserver extremely well.

All updates are done with rdist off of the master system. We are now beginning to look at the bigger problem of distributed backup for our two other types of workstations which don't use fileservers. I hope we can solve that problem as easily as this one was.

Next Generation Planning Tool

Richard Chahley
utzoo!bnr-vpa!bnrnuns!ngpt!rich
BELL CANADA

The NGPT (Next Generation Planning Tool) program is part of a strategic plan to deploy intelligent workstations to access Bell's mainframes. The initial application will be for planning tools which are software programs to assist planners with network configuration studies.

These workstations (initially 240) will be deployed throughout the provinces of Ontario and Quebec. They are IBM PC-ATs running the Santa Cruz Operation XENIX System V operating system. Each workstation is treated as a separate standalone UNIX system operating in a multi-user mode.

Two configurations are being deployed as follows:
Hub: 80386/7 CPUs, 9 serial ports, 2 parall ports, 3 modems, laser printer, 360K/1.2M floppies, 60M disk, 3.5M RAM.
Node: 80286/7 CPUs, 2 serial ports, 2 parall ports, 360K/1.2M floppies, 30M disk, 3.5M RAM.

Each system will have a number of user ids assigned although only one user will likely be logged on at any given time. Five "node" systems will home onto one "hub" system using direct 9600 baud lines to form a 6 system cluster. Within this cluster, each "node" system will send printer requests to the "hub" system's printer and access the "hub" system's modems through the 9600 baud serial links.

Each cluster (approximately 40 in total) is connected to a centralized single workstation (PC-AT "hub") through dial-up 1200 baud lines. It is through this centralized single workstation that system diagnostics, maintenance and administration is conducted for the 240 XENIX systems.

A number of difficulties have been identified with the remote administration of 240 XENIX systems. The following is a (small) list of some of those difficulties.

- End users are required to perform 1.2M floppy backups on the user subdirectory without admin or superuser privileges.
- Console messages appear on the local PC-AT monitor and cannot be remotely viewed. These messages are generally meaningless to the end users.
- Each system's clock marginally drifts. With 240 systems, the impact on cron could be significant.
- Communication failures between node/hub and hub/hub systems are difficult to detect remotely.
- XENIX doesn't allow uucp file transfers across multiple systems. This makes it awkward to uucp common file updates to multiple tandem connected systems.
- Files cannot be uucp'd with the original ownership, group, and permission list in fact. Each file must be located into the proper directory manually.
- Our solution to the uucp limitations is described in the following example.

The system administrator wants to download a revised file which is common to all systems. A shell script uucp's the file to all "hub" systems in the network. Each "hub" is called by the admin system between 1-4 a.m. and the file is downloaded at 1200 baud.

On each "hub" system, cron executes a script which checks for the presence of any uucp'd files. If any are found, they in turn are uucp'd to each of the "node" systems within the cluster at 9600 baud. During this call, diagnostic messages destined for the admin system are transferred to the "hub".

Each "hub" system polls the admin system between 4-7 a.m. and transfers any diagnostic messages from the cluster (hub + 5 nodes); confirmation messages are also sent. These messages are read by the system administrator each morning to verify the transfer.

On every system at 7:30 a.m., cron executes a script which checks for the presence of any uucp'd files, determines the type, changes ownership/group/permission as required and moves the file to the correct location in the file structure.

This procedure has worked well until large files cause contention for the 3 modems.

Software Distribution in a Network Environment

Mike Rodriquez
sun!hplabs!hplmjr!rodrique

The Distributed Computing Center (DCC) of Hewlett-Packard Laboratories (HPL) is engaged in the research and development of software engineering environments, programming languages, artificial intelligence, data-base technology and multi-media interface technologies.

The Technical Computing Environment of DCC consists of two HP9000/840 (Spectrum) machines, three DEC Vax 11/780's, 12 HP9000/200 workstations acting as print/file/mail servers, plus approximately 125 HP9000/300 workstations used by individual researchers. The Spectrum and VAXen machines serve as general time-sharing machines and compute-servers. One of the VAXen is the mail/news hub for HPLabs. The VAXen run BSD 4.3. The other machines are ATT SysVR2 compatible. All of these machines are connected to a LAN via Ethernet. The systems admin group is directly responsible for administering all time-sharing and server machines. In addition we are a resource to aid the researchers in the administration of their workstations.

The problem: Researchers are directly responsible for their own machine's administration. However, they should not be required to perform any more system administration than absolutely necessary. One of the aspects of system administration that we are able to provide a solution for is keeping system files and rapidly changing application files, up-to-date.

We looked into **rdit** as a solution. However it does not allow the user complete control of what files will be installed, and when. (We do, however, use **rdit** to keep the machines under our direct control updated.) To solve this software distribution problem we are using a tool, developed at HPL, called **ninstall**.

Ninstall gives the user complete control over what is installed and when. This utility installs/updates "packages", (collections of files), from a server host. The installation can involve the creation of directories and other arbitrary tasks besides the expected creation/update of files. The installation instructions are maintained on the server host and tell the **installd** daemon/server what to do and how to do it. **Ninstall** establishes TCP connection with the **installd** server on the host specified. Any number of hosts can run the server and provide various packages for installation. Due to server resource constraints (mainly disk space), not all servers provide all packages. Server security is provided via ninstall.{allow,deny} files.

Users run **ninstall** and specify which host they wish to **ninstall** from and which packages they want **ninstalled**. We recommend that users run **ninstall** nightly out of **cron** and specify a standard package that we have developed. Enhancements or bug fixes to the core OS files are also distributed by this package. Maintainers of certain applications with a high degree of volatility have also chosen the nightly **ninstall** method as a means of keeping the users of their software up-to-date.

The philosphy of **ninstall** has evolved over time. Initially it distributed rapidly changing files nightly; now, it also distributes optional software subsystems (is usually invoked interactively). We recently upgraded HP-UX across the entire complex and even rebooted the kernel!

There are some areas concerning **ninstall** where we need increased functionality. One is the ability, from anywhere on the net, to find out what is **ninstallable** and from where. Nightly **cron** entries to run **ninstall** for such slowly changing software are not a good use of resources, and besides, people do not want specialty subsystems changing unexpectedly and without notice. Currently now we rely on mail to notify users when this software has changed.

Ninstall has served us well up to now. Certain of its functions could be replaced with a Yellow Pages type server...the installing of printcap files for example. An area that needs further investigation is the comparison between **ninstall** and NFS. Would remote mounting of a whole mh file-system work as well as having that code accessible from a local disk?

Automatic Software Distribution

Tim Sigmon
Academic Computing Center
University of Virginia
tms@virginia.edu (CSNET)
tms@virginia (BITNET)
...!seismo!virginia!tms

The problem of administering large numbers of networked UNIX systems has been well-documented, as have the increased complications that arise when varying amounts of authority are borne by local system administrators within the network. The Academic Computing Center at the University of Virginia administers a network of over seventy UNIX systems ranging from small workstations to mini-computers.

The systems are grouped into clusters within various departments around the University. Each cluster has a local administrator employed by the local department. Some local administrators are involved, knowledgeable users; some are busy faculty members with little computer experience. The ACC is responsible for developing and maintaining systems software; the local administrators are responsible for system specific day-to-day system administration. The ACC is also responsible for the unenviable task of backing up the more than 10 gigabytes of disk space located on these machines.

When a file should be distributed, many of the machines needing the update are often unreachable and the distributor must remember which machines failed to receive which updates. When problems arise with system software, it is difficult to determine whether or not the machine in question has received all current fixes from the development staff. To address these problems the ACC has written **ru**, an easy to use and simple to maintain system for ensuring that local software updates are distributed automatically. **Ru** functions with minimal effort from the development staff and requires no attention from local administrators. But it allows active administrators choices about which updates and software packages they will accept. Many of the ideas in **ru** are taken from **track**, a system developed at Bell Communications Research [1]

Remote machines maintain a subscription list of software which they wish to keep updated, and may initiate contact with the master machine to obtain those updates. Therefore, local administrators can tailor their software environment by choosing not to receive some software packages or to assume responsibility for their own updates . When machines are down or (as is common for workstations) simply turned off for some period, no harm is done since as soon as a machine is rebooted, it can contact the master machine for any updates that have occurred since the last contact.

Ru is much simpler and less ambitious than **track**. It comprises a few hundred lines of shell scripts. Rather than tracking individual files, **ru** groups software into functional packages and keeps track of one update date for the package as a whole (although only the *new* parts of are transmitted to remote machines). Updates may consist of files to be transferred or of commands to be run either before or after files are transferred. *Newness* is determined by comparing last modified dates for the files in question to the last update date from the remote machine. To distribute an update, the developer places it in a certain subtree and runs a command to set the last update date. The local administrator manipulates a list of packages to which he subscribes and, less frequently, the last date that updates were received for those packages. If the local administrator does nothing, then **ru** quietly updates all packages in a timely fashion. When trouble arises, it is relatively easy (using the subscription list and last update dates) to determine which upgrades have been applied to the machine in question.

---

[1] Daniel Nachbar. *When Network File Systems Aren't Enough: Automatic Software Distribution Revisited,* in **Proceedings of the Atlanta USENIX Conference**, June, 1986, pp. 159-171.

Consulting via Mail at Andrew

Pierette Maniago
pm24#@andrew.cmu.edu

ABSTRACT

CMU and the Andrew project face a few unique issues in system administration. Andrew is a network of high function workstations relying on a distributed file system. Although the creation and installation of the file system is well underway, the application software, including the window manager, are still undergoing rapid change. One of the problems generated by this "experimental" atmosphere is providing user support for this ever changing, rapidly developing system. User support is a two way street. We provide the users with assistance in software usage, file system maintenance, e-mail information, etc. In turn, the users provide us with feedback on the system's performance and usefulness.

Cast of Characters

1: Information Technology Center: Staff of system and software developers and writers.
2: Andrew Systems Administration: Staff of system administrators who deploy Andrew to campus. They also work to insure that Andrew is a multi-vendor system.
3: Academic Computing: User Support, public workstation management.
4: Users: Staff, faculty, students. Currently number about 3800 according to the passwd file. About 1000 are active users of the system.
5:

Because Andrew is still under development, the Information Technology Center is especially interested in the users' problems and perceptions. Feedback from the user community provides the system administrators (SA's) with ideas about what services are needed most. This information helps system administrators prioritize their projects. User support services (a department that deals directly with user problems – this department is often understaffed and sometimes must be bolstered by generalists by necessity rather than by choice) benefits from a closer relationship between the users, the software staff and the SA's. Everybody wins. The challenge is to insure that the interaction between the users, software staff, user support staff and system administrators is as efficient as possible.

For Andrew, part of the solution is a mail handler for a "Dear Abby" account called advisor. Users with non-critical problems and bug reports are encouraged to send an e-mail note to advisor. We've simplified the procedure by providing a pop up menu option that will pre-addressed an e-mail message to advisor. The user fills in the subject line and his question, chooses the send option, and waits. The advisor staff sends a reply within 48 hours whenever possible.

The volume of mail to advisor is very high and varies greatly. The complexity of questions range from ridiculously simple to practically unanswerable with subjects ranging from Unix/Andrew operations to mail networking to sophisticated programming. To simplify the problem, the two person advisor team supervises six students who take first crack at the incoming questions. Each student fields questions from a given day of the week (the weekend is assigned to one student). Questions they can not answer are forwarded to my colleague and me.

How software developers and system administrators fit in

The advisor account automatically reposts incoming messages on a series of semi-private e-bboards according to the sender of the message and the subject line (Refer to pseudo code below). All mail from the user community is reposted to a bboard called advisor.open. The advisor staff (including the six students) subscribe to this bboard and use it as the "receiving area" for new questions. The incoming questions from the users are also reposted to a series of subject specific bboards according to keywords in the subject line. (i.e. if a subject line is "windowmanager bug", the messages is re-posted to advisor.wm.) The method of keyword matching is pattern matching with strings. The keyword string "passw" would match password and passwd. Uninformative or nonexistent subject lines result in the message being re-posted to advisor.misc (also considered a subject bboard). Incoming questions are also posted to advisor.qa and advisor.trail. Advisor.qa will receive final answers and will be converted to a database of consulting information. Advisor.trail also receives final answers and many of the intermediate exchanges. It provides the supervises with an "auditing" trail of advisor activity, allowing us to determine turnaround time and the possible causes of a "dropped" question.

Software developers, system administrators and the consulting staff subscribe to appropriate (according to expertise) subject bboards and to advisor.misc. These "helpers" choose a reply to reader option to post answers, help, and comments back to the subject bboard. If the student staff member was unable to answer a question when it arrived, he collects the input from the subject bboards and formulates an appropriate response to the user. Messages to advisor from those on the "helper" list are posted ONLY to a subject bboard.

Messages generated by the advisor account are bcc'ed back into the bboard system. The answers that the six students and the advisor supervisors return to the users, or any requests for further assistance are bcc'ed to the subject specific bboards. The "helpers" are encouraged to glance over the advisor bcc's to lend assistance when requested and to insure accuracy of translation in the answers that go out to the users. Advisor's outgoing mail is also bcc'ed to advisor.trail. The students and the two supervisors hand cc final answers to advisor.qa.

The following pseudo code describes the method that advisor mail is sorted.

```
helpers = list of uid's of all ITC, ASA and AC staff members
users = not helpers

default advisor.lost                          /*   if not posted elsewhere, put
here.

if ( fromheader   = mailer-daemon ) then
     repost to advisor.open
     repost to advisor.rejections
stop
endif

if ( fromheader   = advisor ) then            /* if from advisor
     repost to advisor.trail
     if ( toheader contains qa ) then
          repost to advisor.qa
     endif (qa)
elseif ( fromheader  = users ) then           /* from user community
      repost to advisor.open
      repost to advisor.qa
      repost to advisor.trail
endif  ( advisor/users )

if ( subjectheader     contains    mail, message, post, send, address, arpa
(etc) ) then
      repost to advisor.mail
      stop
elseif (subjectheader    contains    admin, quota, disk, space, passw, (etc)
) then
      repost to advisor.admin
      stop
elseif ..........
else
      repost to advisor.misc
endif (keyword matching)
```

Electronic Mail Gone Wild

Diane Alter
rtech!diane
Unix System Manager
Relational Technology

At Relational Technology a variety of software and hardware products allow our users to send and receive electronic mail on any one of 20 UNIX and 12 VMS machines. We have 15 other UNIX machines which receive messages, but must be polled to send a message. The Unix systems run 4.2/3BSD UNIX or a variant of AT&T System V. They are all networked together via a mixture of DECNET, TCP/IP ethernet and hard-wired serial lines to our Bridge Communication Terminal Servers.

Many of our engineering users receive their mail on UNIX but the majority of engineering and marketing home accounts are on VMS. The number of mail messages we deal with is phenomenal. In addition the Data Center maintains over 90 company-wide electronic mail lists. Some of these lists contain over 200 names. These lists are maintained with shell scripts on UNIX.

One of our DEC MVAX2 (RTECH) running Ultrix 2.1 is the center of this electronic message network. It has DECNET and TCP/IP ethernet and the ability to gain access to our Terminal Servers. The Terminal Server allows any terminal attached to it to connect to any machine we have in-house by means of establishing a temporary virtual circuit. By means of an entry in /usr/lib/uucp/L.sys for each local machine, UUCP connects via the switch to the any local machines. This machine is also our link to the outside world. It is able to send and receive uucp and mail messages to and from outside companies, as well as call up our Netnews feed.

The heart of the mail routing scheme is the /usr/lib/sendmail.cf file on RTECH. We are fortunate to have an employee who worked on changing sendmail.cf for The UUCP Project, and he tailored our network layout into sendmail.cf. He wrote two different sendmail.cf files. RTECH's sendmail.cf knows how to send to local and outside uucp sites as well as use the mail11 mailer to send to our VMS machines over DECNET. It also transforms UNIX mail header syntax into the equivalent VMS mail header format and vice versa. The other sendmail.cf file is used on all the other TCP/IP ethernetted UNIX machines. It handles the local UNIX-to-UNIX mail delivery and hands any uucp or VMS mail to RTECH to deliver. Both sendmail.cf files have statements which include three static files of local machine names. Each time the configuration file is refrozen, the contents of the files are incorporated into sendmail. The only hardcoded machine name in the file is our outside world name RTECH.

Along with /etc/hosts we maintain four other host related files. Each time a new machine is added to our network, the master copy of these files are updated and distributed to each networked UNIX machine. Next, the sendmail file is refrozen and the daemon restarted. All commonly updated files are copied into the /etc/dist directory on each system. These files are actually symbolic links to the location of the real files. This common directory allows for easy updates and restores each time the OS is upgraded or /etc is scrambled.

Another key file for the support of the mailing lists is /usr/lib/aliases. Each mailing list is defined as two other aliases. One alias is the expanded list of UNIX users and the other is an alias on RTECH to the VMS users on the list. The VMS alias is not the a list of user names. Instead it is an include statement of a file containing the VMS users and their home machines. These lists can be changed and used without rebuilding the aliases database.

On VMS, the users use a logical which points to two files replicated on each VMS system from the master mailing list file on RTECH. One file is the list of VMS users and the other a one line forward to the list of UNIX users which is in the aliases database on RTECH. This is the same list used by UNIX users who send to the mailing lists. This way only one mail message is sent to RTECH which is expanded to all the UNIX users on the list and then sent out. This dramatically reduced the amount of mail sent between UNIX and VMS.

Now each aliases database on the UNIX machines on which we allow home mail accounts is unique. They all have the common mail list aliases, but many have aliases particular to that machine. For example, we have aliases on RTECH for anyone in the company who wishes to receive outside mail. Outside persons need only send a message to rtech!username and the message will be sent to the user home UNIX or VMS account. These files are updated periodically with new mail list names. To handle these mass updates on all the systems a login was created on each of these machines which runs a shell script to login into RTECH, generate the new aliases file (from the mail list files) for that machine, move it to the machine, and then run newaliases to rebuild the aliases datebase.

It is amazing that all this software hangs together with minimal maintenance. Mail list changes and distribution to UNIX and VMS machines are done nightly by our lead operator. Also the logical UNIX:: is setup on each VMS machine which equates to RTECH. VMS users use this logical to send messages to a UNIX user. It was done this way so that if RTECH went down for a long period of time, we could change the logical to another MVAX2 to do the routing of local mail without a long disruption of service.

But the system does have it problems and shortcomings. Users have the ability to change their forwards and they don't always do it correctly. On UNIX they setup their forwards to thrash back and forth between two machines until eventually the messages times out due to excessive hops. On VMS they wind up exposing the known VMS bug regarding multiple forwards to different machines which hangs VMS mail. Another problem is that our machines which meet SVID and have TCP/IP ethernet can not use our generic sendmail.cf file. Until the problem is resolved they have not been included in this bulk update process.

One shortcoming we have not resolved deals with our remote machines. Currently two of our VMS machines are located in London. They are connected to our local DECNET network via a dedicated undersea 9600 baud twisted-pair phone cable. Future plans include adding our PARIS and SYDNEY offices to our network with similar lines. We would like to have an alias for the main London machine be "london", so that users do not have to figure out which machine in London an employee receives their mail on. VMS handles this nicely with the use of logicals, but on UNIX neither mail11 or DECNET can handle multiple names for a given host. Our best solution so far is to add a wrapper to mail11 to allow multiple names for a host. I don't like this idea because I don't like to embed such programs (or scripts) in low level routines or daemons. I am looking for a better alternative.

Using News Multicasting with UUCP

Dr. Rick Perry
{cbmvax,pyrnj,bpa}!vu-vlsi!perry
perry@vuvaxcom.bitnet
Department of Electrical Engineering
Villanova University, Villanova, PA 19085
215-645-4224 wk, 215-259-8734 hm

The Electrical Engineering Department at Villanova University felt the need for computer resources beyond those currently available from our computer center and shared by the whole university community. Thus, about three years ago, we purchased a Pyramid 90x machine and set up a VLSI Design Laboratory running the Berkeley CAD tools. Since that time, the usage of the machine has expanded so that most of our faculty and graduate students use the Pyramid for their research computing, and our network connections have grown to include most of the other local Philadelphia area Universities. We currently send Usenet news feeds to four other sites.

We have two Fuji Eagle disk drives and (through extensive use of compression on seldom used files) have plenty of free disk space available. But there are times, especially when a lot of news comes in at one time and gets batched to be forwarded downstream, that our /usr/spool partition gets very close to overflowing. These large news batches also cause hassles sometimes when our daily incremental backup procedure gets going and there never seems to be enough tape around to hold everything. So I decided to take advantage of the MULTICAST option in 2.11 news and set up a news batch forwarding procedure to minimize the spool load.

For each of the four sites which we feed, I changed the news sys :F entry to :M:multi and defined a fictitious system 'multi' with a :F field, so the names of news files destined for any combination of the four sites get written to /usr/spool/batchnews/multi in preparation for later batching. Let's call these four sites 'a', 'b', 'c' and 'd' for discussion here; lines in the multi file then typically look like:

```
/usr/spool/news/comp/ai/234 a b c d
/usr/spool/news/talk/misc/123 a b c
/usr/spool/news/sci/space/453 a c d
```

This situation illustrates a case where systems a, b, c, and d all subscribe to comp.ai, but d does not get talk.misc and b does not get sci.space. For each possible combination of system names from the set {a,b,c,d} separate news batches will have to be created.

So what we do is copy the multi file through sed, substituting a colon for the spaces between system names, and the resulting file, multi.tmp2, looks like:

```
/usr/spool/news/comp/ai/234 a:b:c:d
/usr/spool/news/talk/misc/123 a:b:c
/usr/spool/news/sci/space/453 a:c:d
```

The set of unique combinations is then easily generated via:

```
NAMES="`sort +1u multi.tmp2 | \
    awk '{ print $2 }'`"
```

and for each combination we create batches of the associated news articles.

All of this would be a waste of time however if it was not possible to queue uux commands that did not make a separate copy of the batch file for each system. Fortunately, Pyramid's uux is based on BSD4.3 and has the -l option by which it creates a link to the file instead of copying it. So, assuming the news batch program output is in a file batch.tmp, the following sequence of commands queues the files and removes the original:

```
 for rmt in `echo $combo | \
   sed 's/:/ /g'`
 do
   uux -l -r -n -z -gd \
      $rmt!rnews\< !batch.tmp
 done
rm batch.tmp
```

The Bourne shell script to do this whole thing is available from me by mail.

Electronic Mail Maintenance/Distribution

Yoon W. Kim
yoonkim@Sun.COM
Engineering Systems Support
Sun Microsystems, Inc
Mtn. View, California

There are 3500 plus employees using over 2000 workstations networked here at Sun Microsystems, Inc. Our workstations are based on 4.2 BSD UNIX and its networking environments. All the workstations here at Mtn. View are connected with ethernets and the remote sites such as sales offices around the world are joined to this main network through internetwork routers (INRs) over high-speed serial lines.

Throughout the company the electronic mail (e-mail) system serves as a main communication channel. And it is, needless to say, quite a task to just keep the system flowing smoothly at all times. Within the company, keeping track of all the employees' e-mail address becomes the first priority.

The employee's e-mail address is turned into a standard form which is the first-initial plus the last name. In case this form is ambiguous we use the full first name or some combination thereof to make it unique. And if anyone uses (i.e., sends mail to) the ambiguous alias, a reply is sent back automatically with a list of possible aliases. This standard alias is, then, mapped to the real employee's login name at his machine name in the form of *user@machine*. So, mailing to *ykim* would eventually be resolved to *yoonkim@narnia* and arrive in my mailbox. Mailing to *yoonkim* (the user's login name as oppose to his standard alias) will first be resolved to *ykim*, then to *yoonkim@narnia*. This all works fine if every machine had this aliases file, which you can imagine how big it can be. Distributing such a large file is not only inefficient, but also unreliable. So we use the Yellow Pages (YP) database to propagate this aliases file around.

This enables each machine in a YP domain to know all the employees' aliases right on that machine without having to rely on a "mailhost". Before, if you didn't know the exact e-mail address, you had to pass it on to a mailhost which had this master aliases file and would resolve the alias and pass it on to the destination. So such a mailhost became a bottleneck in a hurry. But now with YP aliases, each machine is able to resolve (even the large mailing lists) to the final destination in a single hop. For routing to other networks such as USENET and ARPANET, a mailhost is set up to handle this. So if a machine doesn't know how to route to another network, it will push it over to the mailhost where it will be routed through or returned to the sender.

This idea of every machine having YP aliases has one drawback with expensive high-speed (INR) links in Sun's network. Let's say side A and B are connected through this expensive link. And if someone on side A sends to a mailing list which contains 100 users on side B, then sending 100 messages across this link is quite costly. It would be much more efficient to send over one message and let a sub-mailhost on side B send out to 100 users locally. So such a setup is available on each side of expensive links.

This domain-wide aliasing through YP databases capability saves lots of e-mail address lookup time by users, and offloaded one mailhost processing all the mailing lists thus speeding up the delivery time as well. As far as the mail system is concerned, all the workstations are integrated in a single domain with this method.

User Account Administration at Project Athena

Janet Abbate
abbate@athena.mit.edu
Systems Support
Project Athena, MIT

Project Athena's goal is to integrate computers into the MIT curriculum and environment, using hardware supplied by DEC and IBM and BSD 4.2 UNIX. Currently there are about 400 VAX 11/750s, VSIIs, and IBM RT/PCs, but that number is scheduled to increase to 1500 over the next year.

In a project this size, maintaining user accounts poses a major challenge. The user community consists of the entire MIT undergraduate population (several thousand students), some graduate students, and those faculty members who choose to develop course materials on Athena. We face the problem of keeping user information such as passwords consistent from machine to machine, and of making this information easily accessible to inexperienced users.

The solution at Athena was to create a central database of user information. The database is implemented in RTI Ingres and contains data on our users, courses and projects, clusters, and mail aliases. Information that must reside on the local systems, such as password files and mail aliases, are propagated from the master system several times a day. The entire database is backed up to a different machine each night of the week, so that seven different systems can serve as backups in case the primary system fails.

For security reasons, the database resides on a restricted machine and can only be accessed directly by privileged users. Users and administrators access and modify the data through various utility programs. The interface between the central database and the user programs running on local machines is taken care of by a daemon (called "userinfod") on the master machine. Programs also create user accounts, maintain mailing lists, change group affiliation (used by course administrators and project leaders), change root passwords, and backup the database. They provide a convenient interface for the users (menus) and allow group administrators to maintain control over their own accounts; besides relieving the system managers.

Students receive Athena accounts upon entering MIT and the accounts remain active throughout their undergraduate years. Before the start of each semester, a list of new undergraduates is read in from a tape and added to the database. As the semester begins, these students "activate" their Athena account by running a program that verifies their identity and has them choose a password for their account. This takes care of the vast majority of students. Accounts for course work are assigned by groups; all students in a given course are assigned to that course's group and are then able to access the course account.

In addition to user accounts, the database keeps track of Athena's superuser accounts. To keep the number of root passwords at a manageable size, each user cluster is assigned a single root password for all of the machines in that cluster (development machines are more secure and tend to have unique passwords).

There are still problems that remain to be addressed. It is much more difficult to remove users from the system than it is to add them; it is possible to "deactivate" user entries in the database, but it has to be done by hand, and removing the users' files from the system is a separate step. Also, the system depends heavily on a single machine and would be painful to recover if that machine had problems. Future plans for Athena include integrating the user data into a comprehensive "Service Management System" that will provide a central database not only for user information but for name servers, printers, and other services.

An Automated Student Account System

Lloyd W. Taylor (lwt1@aplvax.arpa)
John R. Hayes (aplcen!john)
Johns Hopkins University/APL Center

The Johns Hopkins University runs a Part-Time Graduate Education program that leads to M.S. degrees in Computer Science, Electrical Engineering, Technical Management, Numerical Science, and Applied Physics. Of the 1800 students enrolled, approximately 900 are pursuing degrees in Computer Science. Most of the students are professionals employed locally.

In the summer of 1984, a Computer Center was installed to support the growing enrollment. Unix (System V) support is provided on an AT&T 3B20S computer, donated to the University by AT&T. VMS support is provided on a VAX-11/780, donated by a private individual. An Equinox Data PBX hooks everything together.

The problem of account creation, deletion, and maintenance is phenomenal. We create and delete over 650 accounts each semester. Students with continuing enrollments must retain their files between semesters, and instructors insist that each of their classes be in separate groups and in contiguous directory trees. The administrative load to maintain this system would be staggering were it not for the *newuser* system.

At the beginning of each semester, and periodically throughout, we obtain from the registrar's office a IBM-format tape containing the entire student database. This tape is read into the system and assembled into the resident student database. An index file is created to speed access to the 2000 or so records. A database, called the *classlist*, contains a list of classes eligible for accounts on the 3B20.

To apply for an account, a student logs in to the 3B20 as *newuser*. The system requests the Social Security number (hopefully a unique identifier) and presents the student with a list of allowed accounts. The student may apply for separate accounts for each eligible class, or may use a single account for all.

If this student has not had an account before, the system prompts for a desired userid and for the student's phone number. This information is then entered into the resident student database and an account request is submitted. If the student had an account the previous semester, it can be reused. Old files are restored.

Early each morning, the system operator runs a script that creates all requested accounts. The system generates pseudo-random passwords, and prints out account sheets that are given to the course instructors to pass out to their students.

When a new registration tape is received, it is read in and compared against the old data. Section changes are handled automatically. Course drops are handled manually, to ensure that no student's account is deleted due to an error in the registrar's records.

Thirty days after the end of each semester, all student accounts are locked out by substituting a lockshell for the standard Korn Shell. When a student re-applies for an account the following semester, the lockshell is changed back to the Korn Shell, reauthorizing the account. Five weeks into the new semester, a script to remove users is run that deletes all accounts that still have the lockshell.

Performance has been excellent. Account maintenance takes no more that a couple of hours at the beginning of each semester, and very little time during the semester to handle account generation. The system is handled by relatively untrained operators, who have had little trouble understanding the procedures.

In addition to minimizing the administrative load, there are a couple of additional advantages. First, only those students who desire accounts receive them, thus keeping the size of the passwd file down. Second, students can pick their own username, making the system a little more personal.

Academic Computing Services and Systems (ACSS)

Marshall M. Midden
(m4@umn-acss-ux.arpa [128.101.63.2])

Academic Computing at the University of Minnesota is provided by eight cooperating computer centers. [The Minnesota Supercomputer Center provides the appropriate support needed for that aspect of computing (including the U's physical Arpanet connection).] There is a campus ethernet that connects most of the Twin Cities campus along with gateways going to the supercomputers and the Duluth campus.

ACSS's role is to provide computing support for academic programs of the Twin Cities campus. Crookston, Duluth, Morris, St. Paul, and Waseca also provide their own academic computing support. Administrative computing is a separate identity. Health Sciences provides support to the University hospitals. In addition, many departments in the University manage their own computers and LAN's.

ACSS handles a large local area network that provides links from a large part of approximately thirty labs (Twin Cities campus) with over 250 terminals, 350 micros, and various workstations to currently four of the academic computer centers. ACSS has a CDC 855 (NOS), CDC 830 (NOS), VAX 8600 (VMS), and an ENCORE MultiMax (UNIX). The organization runs a computing information center on campus, contract services group, user services group, 8-5 help line (626-5592), microcomputer group, engineering (field) services, in addition to networking, system programming, accounting, operations, publications, applications and tools, languages, and some support in public labs including a shuttle service between the various locations on campus. (ACSS has its computers located about six miles from the main campus in the tiny suburb of Lauderdale.)

In January of 1986, what was the University Computer Center was fissioned into two separate departments, the Minnesota Supercomputer Center (MSC) and the Academic Computing Services and Systems (ACSS). ACSS just recently got back into the Unix world. The Encore computer currently has 96 ports, two ethernets, 8 processors (0.75 MIPS each), 16 MB memory, seven 415 MB CDC disks on two SCSI buses with four disk controllers, and two tape drives. We ordered two 200 IPS start/stop tape drives, but due to difficulties only one 75 IPS streaming drive was delivered and just recently a second one was attached (two tape controllers and formatters). The system is running ENCORE's own version of 4.2BSD; we just recently succeeded in getting the paperwork for source.

There are about 2300 login names, and due to compile time parameters in many programs (we used to have a Vax 11/780 running 4.2 then 4.3BSD) it takes several minutes to do such things as "ls -l" on certain directories. The header files are not completely standard – mostly due to the file system not being Berkeley standard and some minor changes to the ethernet stuff. In addition, many of the programs in /bin came from the companies original System V implementation – "with additional parameters and functionality added." Most of the 4.2 programs that need to be fixed reside in /bin and have not been able to be fixed locally (or 4.3 BSD upgraded). The "mail" and other networking features of unix have been on standstill due to many problems and slow correction time from the manufacturer.

GOOD NEWS though! The users really like the responsiveness of the machine. We can hang 6 of the 8 processors in fatal loops, and the classroom students hardly even notice it! (It is necessary to clean the system process table daily.) We would rather NOT have most utilities multiprocessed. This makes it more difficult for a user to get (and hang) more than one processor. (The Berkeley implementation multiprocesses across pipes.)

For classroom accounts, we have implemented a program that creates sequences of account names (such as 5477m00 through 5477m99). In addition, a mail alias is created for the instructors, a possible separate group, and a master user number is created to oversee the class. At present the master user can "su" to any of it's subordinates, or "chmod". Due to the size of some master users and the slowness of 4.2's password related subroutines, a rewrite is underway to speed it up. (It will be late this year before the company has projected 4.3BSD to be available. A very strong request for a more compatible port has been made.)

File systems are set up as one disk drive. Not counting root, tmp, and a spare disk drive, the disks are running at 62% used. Operation's staff do file system dumps: a full weekly, long daily (11pm) [to last full], and shorts twice daily (1pm, 6pm) [to last full/daily]. Full and daily are to separate tape sets, and shorts have all the file systems on one tape. No file archiving for unaccessed files is currently done. Full dump tapes are alternated between a rack and the vault, with the "first of the month" full dump going to another another rack, where it is saved for three months. Also saved is the last dump from the end of fiscal year. Tape usage for one full dump set is 14 tapes for the Unix machine, and 89 for the rest of the machines (6250 BPI). Due to volume, no tapes are kept off-site. (Halon system, very large machine room – tapes kept at opposite corners of the room)

We are considering Reel Backup and Utilities for next fiscal year, although it doesn't fit schemes used on our other machines. (On our Cyber 855, we have 8000 tape mounts per month – down from 12000.) Any other tape management schemes available?

Password File Management at the University of Maryland

Pete Cottrell
pete@mimsy.umd.edu
Department of Computer Science
University of Md.

The Department of Computer Science at the University of Maryland maintains its own computing facilities, separate from those maintained by the Computing Center for educational purposes. The CS Department's machines are used for research by the professors and graduate students, although a limited amount of class work is done on them. Our equipment consists of 2 Vax 750s, 1 VAX 785, 1 VAX 8600 (all running 4.3BSD), a Pyramid 90x (version 2.5 of their dual Unix port), a Data General MV/10000 (running their newly released DG/UX 3.10), several laser printers and 55+ Sun workstations (level 3.0/3.2 of SunOS). In addition to these, there are approximately 38 Xerox workstations and associated servers and laser printers available. The machines are contained in 2 different buildings and all of these are on the same ethernet that stretches between the buildings. We have 4 full time staffers (one dedicated to hardware) and about 12 student workers.

When our facility really got started, we had 3 of the Vaxes and the Pyramid. Accounts were put up with no concern for whether or not the person had the same uid on the other machines, although we have always kept the user names unique (i.e., there are not 2 different people with the login name 'pete'). Then we started getting the Sun workstations with NFS; the ability to access files on other systems was the obvious motivation for going to a unified-uid scheme. By the time we were able to start the project, we had added the Data General and the 55+ Suns, none of which had any consistent uid numbering.

We also decided that we should unify the group ids (gids); again, this was done to accommodate NFS. We decided that if someone was in a group on one machine, they would be in that group on all the machines they had accounts on.

The problems facing us were:

- How to reassign the uids/gids (IDs)
- How to change the IDs of files once we had done the reassignment
- How to do all of this with a minimal impact on our users.

We also saw the chance to reclaim some disk space and tidy things up by identifying all of the files owned by stray IDs. Most of these files were the result of software coming from other sites or were never cleared when their associated account was deleted.

Chris Torek wrote 2 programs that did most of the work; they are called 'mp' and 'zap'. 'mp' takes as input the password and group files of the machines you want to unify. It will also read in base password and group files in which you may explicitly assign various IDs to different users and groups (e.g., 0 is assigned to root, 2 to bin). When run, 'mp' takes its data and reassigns IDs, producing for each machine a map file. This map file contains records specifying the old and new uids for a user. These map files are in Internet network byte order, so are transportable across different systems. 'mp' also produces a new password file for each old one provided and produces a single master group file that can be used on all machines.

Then 'zap' is used for 2 jobs. When first used, 'zap' identifies all of the files owned by stray IDs. This gave us the chance to do something with these files. Most were deleted, while the others were assigned a proper ID. Once all of the errant files were taken care of, 'zap' was run with the '-doit' option; this causes 'zap' to go out to the raw disk and use the map file to assign new IDs to the disk inode uid and gid fields.

The procedure used is fairly simple. We collected all of the password and group files and ran 'mp' to produce the new password and map files. Then, for safety's sake, we would dump a machine before 'zap'ping it. After dealing with the stray files discovered by the first run of 'zap', we shut down the machine and ran 'zap' with the '-doit' option. 'zap' runs very quickly, so down time is kept to a minimum; the dump and zap time for a Sun with a local 85meg disk was usually about 20-25 minutes. Then the new password and group files would be put into place and the machine would be rebooted. This scheme allowed us to do one machine at a time without affecting others.

The whole process took about 2 weeks, but could have been done in a shorter amount of time. Some of the problems we encountered were:

- access to the machines. Scheduling a time to get to a professor's machine was difficult at times.
- No new accounts could be created during this time because they wouldn't be contained in the map file.
- We quickly realized that any files indexed by uid number, such as lastlog and the quota files, would also - need to be remapped. Programs to do this were quickly whipped up.

As for the code itself, it was extremely portable (I remember only one fix being made) and ran without incident on 4 different makes of machines. The only real scare we had was when we ran 'zap' on the 8600. After moving the new password file into place, a staffer did a few 'ls's to see if things had been done correctly. Unfortunately, we had failed to run '/etc/mkpswd' to remake the hashed password database, so the long option of 'ls' showed some strange IDs. This was easily dealt with.

Now we are dealing with the administrative problems created by this project. New accounts have to be assigned a unique uid and group updates have to be written to the master group file. This file has gotten too big, so we wrote a program called cleangrp that 'cleans' it out. Each night the master group file goes out to the individual machines and 'cleangrp' is run, removing the names of uses who don't have accounts on that machine. Deleting accounts is more work too; if a user is having his last account deleted, then we have to be sure to delete him from the master group file and then mark his uid as available for reuse. We have also written a variety of other shell scripts and programs to do similar administrative things, such as consistency checkers. We are always looking for any thoughts and ideas on how to manage accounts in a distributed environment.

If there is sufficient interest, we will offer our software to other sites.

Hacct - A Charge Back System

Stephen Uitti
suitti%husc8@harvard.harvard.edu
Sr. UNIX Systems Manager
Science Center
Harvard University

The Harvard University Science Center runs five uVAX II's, a VAX 11/780, a VAX 11/750, and a PDP11/84. The 750 is owned by the mathematics department but run by Computer Services in the Science Center. There are five uVAX II's run separately by the departments of Sociology and Psychology. Additionally, a pair of phototypesetters are run on a cost per use basis; some research use and commercial use is provided. Also, the services of technical typists, secretaries are timeshared, and must be billed separately. To achieve this, the Hacct charge back system is under development.

The new system should provide the following facilities:

- Users should only have to have one account.
- Users doing work that needs to be billed separately should be able to switch between billing entities easily and quickly.
- The system must maintain a single set of charges across machines in a Network File System environment.
- Users should be able to use any hardware resources available (subject to political restrictions only), including any of several CPU's, etc.
- System Administrators should be able to change access control and billing rates for a variety of resources on a per billing entity and per resource basis.

Technical issues have also shaped the design of Hacct. For example, UNIX lacks a reliable file system record locking mechanism (especially in an NFS environment). The solution is to have only one process do database file manipulation. BSD UNIX networking is then used for interprocess and interprocessor communications.

The design has also been affected by past (and present) experience with other accounting systems. Harvard's "old" accounting system has some serious deficiencies: it is hard to port due to kernel modifications, and many (many) user level modifications. When the database is corrupted, it can be difficult to correct, and difficult to access the system (just to correct the problems). It makes the password file subservient to it, and replaces the group file altogether. This makes porting new software (that appears, for example, via Usenet) difficult. Additionally, recompilation of most of the software is required when it is desirable to add or subtract even a single field to/from the database.

The new system, Hacct, is designed with the idea of minimal impact on the operating system. There is one optional kernel modification (for efficiency). Login (1) is modified to allow users to log in as user.group pairs. A very few utilities are changed to grant and refuse resources (e.g., login(1) and lpr(1)). The Hacct database does not replace, duplicate, or change the format of any information contained in standard configuration or log files. For example, the /usr/adm/wtmp file still records login/logout information. Group information is added to it by having login(1) post the usual user ID entry, then a "newgroup" entry. The newgroup entry is not a new format, just a new use of an old format.

We've done some work on having a configuration file drive the field specification in the database. There are currently two main configuration files - one which specifies the fields, the other which specifies the rate structure. The goal is that with some constraints, fields may be added at will, without requiring modification or recompilation of any utilities. At the very least, the addition of new resource types and modifications to the rate structure will be possible on a "live" production system.

Login Management for Large Installations

Evelyn C. Leeper
ihnp4!mtgzy!ecl
AT&T, Middletown, New Jersey 07748

I am a system administrator for a very large shop: it includes twenty-one machines, running on four different machine types and three different versions of the UNIX operating system (UNIX is a registered trademark of AT&T). We support over 1000 users, with our largest system supporting almost 300 users. A core problem is managing the password file and various related files.

Adding a User

Every new login requires that the user fill in a request form and have it signed by his or her supervisor. This is true whether it is for a new user or an existing user to be added to a different machine. To add the login the operator runs an interactive program, which requests all necessary password file information. In our case this includes department and supervisor as well as the standard password file entries, for the reasons explained below. For many of the entries (e.g., group ID) the program offers a default option calculated from previous answers or currently existing logins. After the entry process is finished, the program:

- updates **/etc/passwd**
- updates the files that are used to decide the CPU allocation under the Fair Share Scheduler [1]
- creates a login directory for the user and an rje directory (the default delivery directory for all files sent to the user via the JES3 network)
- sends mail to the people maintaining the on-line personnel and project databases asking them to update those databases

This is all just one step and avoids potential errors in editing.

In our networked environment, one person's login name and user ID need to remain constant across systems, for practical reasons (i.e., there should be a bijection between people and login names and between and usr IDs). We keep copies of all password files in a special directory on all systems. The program to add users verifies login names and user IDs against these files. This also provides us with backup passwd files in the event of system problems.

Tracking a User

The organizational information included in the password file allows us to build a hierarchy of logins. This hierarchy helps track resources that are distributed by organization–fair share scheduler allocations, disk allocation, etc. Now it can be determined when new logins would duplicate old ones. Because supervisors realize that their allocations are divided among all the logins they authorize, they do not automatically approve every request. It is also easy to see which organizational entities are consuming the most resources so that adjustments in allocations can be made.

We also run a series of tools daily which check the consistency of the system files. Tracking programs can be run easily because of the password file database stored on each system.

Deleting a User

Since all the password files are available on each system, deleting a user is easy. It does not require that the system administrator log onto each system to check whether or not that user has a login.

We have written several automatic tools to handle clean-up after a user is deleted. For example, one tool checks the mail spooling directory for files not owned by a valid user as well as for old forwarding files. Also, we check the common areas for files not owned by valid users and see that they are deleted or have their ownership transferred to a valid user. This latter program avoids having disk space used up through attrition.

[1] Henry, G. J., "The Fair Share Scheduler", *AT&T Bell Laboratories Technical Journal*, October 1984.

Sharing Accounts

Matt Bishop
mab@riacs.edu
Research Institute for Advanced Computer Science
NASA Ames Research Center; Moffett Field, CA 94035

(Work reported here was supported by NASA under contract NCC2-387.)

The Numerical Aerodynamic Simulator project runs a variety of UNIX based operating system, on its computers (a Cray 2, 2 Amdahl 5840s, 4 VAX-11/780s and 25 IRIS 3500 workstations.) Users work on and off site, using a variety of networks not all of which are under NASA's control. Off site installations can be as close as a different building on the base or as distant as ICASE (on the East Coast.) In our environment, sharing accounts is very common; it provides a very quick and easy way to enable many people to work together, and allows many people to share responsibility for a particular task. For example, the account *nasops* is used to maintain a database of users (among other functions.) So, many people need access to that account. Unfortunately, this poses some problems.

First is the question of accountability. If someone logs in on the account *nasops* and compromises the database, how can the offending user be traced? Password management is the second problem; how can the site administrator force 40 or so people to keep the password secret, particularly since these users need at least two passwords (one for their own account and one for the *nasops* account)? When someone changes *nasops*' password, how does he or she communicate that change to the other users in a timely manner?

A group account is an account meant to be shared. No-one can log into a group account, but an *su*-like program called *lsu* overlays the login identity with the group identity (just as *su* overlays the login identity with a new user's identity.) The user must type his own password when switching to the group identity. *Lsu* checks an access file to ensure that the user can access the group account at the given time and from the given terminal, and then checks the password. If access is allowed and the user types his own password correctly, the group identity is pushed over the user's identity; if access is denied or the password is incorrect, *lsu* simply informs the user permission is denied.

Because of the sensitive nature of the program, we took several steps to prevent compromise. While it seems redundant to require the user to type his password (didn't he type it to log in?), experience shows that people do leave their terminals unattended, so checking the password provides some assurance the user is the person running *lsu*. The password must always be typed, even when *lsu* has already determined the user has no right to *lsu* to the group account. The location of the access and log files are constructed in memory when *lsu* runs, and are erased immediately after being used. Both files must be owned by *root* and must be mode 600; if not, the *lsu* fails and mail is sent to the *lsu* administrators. Of course, when access is denied, the user is not told why access is denied.

We also intend to provide the same control for overlaying user accounts. The program *nsu* functions like *lsu* but requires the new account's password as well as meeting any conditions in the access file. (If the account is not listed in the access file, anyone can *nsu* to it.) In the event the access file is trashed, *nsu* can only be used to access *root*.

Lsu and *nsu* use the startup file of the new (group or user) identity, not that of the user running the program; the environment variables USER and HOME are also changed. This provides uniformity among users who may have wildly different environments in their private accounts. A third program, called *su*, provides the usual *su* environment but uses the *nsu* access file to check permission.

Currently *lsu* and *nsu* run on twelve different systems (Berkeley's 4.2 and 4.3 BSD, Sequent's DYNIX 2.0, the NAS' NPSN 3, Ridge's ROS 3.3, Silicon Graphics' SGI 2.3, 3.4, and 3.5, Sun's 4.2, AT&T's System V, Amdahl's UTS, and Cray's UNICOS.) User reaction to *lsu* has been very favorable, largely because of the consistent environment and the relief from remembering multiple passwords. *Nsu* and *su* have just recently been made available, so we do not yet know how the user community will accept them.

Creating an Environment for Novice Users

Jeffrey M. Smith
aat@j.cc.purdue.edu
Purdue University Computing Center

The Purdue University Computing Center (PUCC) supports mainly "instructional" computing on its network of 3 PDP-11/70's, 6 Vax 11-780's, and Sequent Balance 21000. These Unix hosts support thousands of users, mostly students in Computer Science and other courses.

Learning Unix can be made much less painful than it normally is by creating a custom environment for novice users. In the past, each user's environment was created (if at all) by the course instructor. Typically the instructor would have students run a shell script that would copy a standard set of "dot" files to the student's account. However, instructors often do not wish to take the time to create an environment, and don't. When they do, they sometimes make mistakes that are detrimental both to their students and system resources. Also, the standard scripts don't take the user's level of experience into account when creating an environment.

In response to these problems, PUCC has developed an "enrollment" shell, *esh*(1l), which is the initial login shell for most users. *Esh* is an interactive, configurable shell that creates an environment during the initial login and then replaces itself with what will become the user's permanent shell. By interacting with the user, *esh* can modify its actions according to the experience of the user. Since *esh* is easily configurable, there is more incentive to tailor custom environments for homogeneous groups of users.

Although *esh* was specifically developed for the instructional environment at PUCC, it is easy to imagine its use in other situations. For instance, a business might have a secretarial pool that would require a special text processing environment, a group of accounting clerks that would need an environment tailored for using speadsheets or maintaining databases, etc.

Esh uses a configuration file of questions created by the systems administrator. It can take arbitrary actions based on the answers it receives (this is how the user's experience level is taken into account. For instance, if a user claims to be an experienced user of the news software, a ".pnewsexpert" file might be created.) Running as the user, *esh* executes shell commands based on the questions it asks and the answers it receives. Typically *esh* creates a ".login" and a ".cshrc". Often a .newsrc will be created, subscribing the user to certain groups that the course instructor wishes him or her to read (at PUCC, we encourage instructors to use news to communicate with students instead of mail.) Often a ".mailrc" or ".mh_profile" will be created.

Another advantage of using *esh* is that we are able to create an environment that lessens the user's impact on the system. For instance, we are able to head good-intentioned but misguided instructors off at the pass, as when an instructor set up ".cshrc" files for students with the line "set mail = 5"! Each student's mailbox was stat'ed every five seconds (and of course *biff*(1) would have given instant notification with very little overhead).

Esh has met with a positive reception both students and from course instructors, who specify the type of environment they wish the student to have. In fact, students who registered for the second half of a two part course were dismayed when they found that the instructor had failed to specify *esh* as their initial shell.

Esh helps the new user get accustomed to Unix more quickly, while taking his or her level of experience into account. It allows instructors or supervisors to create an environment with certain standard features that they find desirable. Finally, by involving the systems administrator and programming staff, it allows PUCC a degree of control over user environments.

PUCC would be happy to make *esh* available to anyone who is interested in obtaining it, either via anonymous ftp or posting the sources to Usenet.

Priv: An Exercise in Administrative Expansion

Eric Heilman
heilman@brl.arpa

The responsibility for administering and maintaining the Ballistic Research Laboratory computer resources is the task of the Advanced Computer Science Team in the Systems, Engineering and Concepts Analysis Division. These facilities include a number of Sun work stations, several Gould and Dec minis, CDC Cyber mainframes and a Cray supercomputer. Each of these individual machines is interconnected by several local area networks ranging from a microwave link to a 10 Mbps Proteon fiber optic ring. Sensitivity of the research material existing on this network requires the number of operators with "superuser" password access to be limited. Unfortunately, the daily maintenance needs of this extensive, user community exceeds the capabilities of the dozen administrative "superusers". Rectification of this problem was not achieved by establishing a special group which could perform system tasks such as file system backups, shutdown and MOTD modification. Therefore, a new systems utility was deemed necessary. This utility, called Priv, created "pseudo superusers" that expanded the administrative operator body and alleviated the administrative bottleneck without jeopardizing the security of ongoing sensitive research.

Written in C, Priv permits the "pseudo superuser" two methods of accessing a special subset of system maintenance commands. Known as "Priv.conf", this configuration file allows limited administrative file manipulation (but not errant program or file modification). Specifically, the special operator may invoke a user name or may take a root system shortcut to assist the user community.

In the first method, Priv accepts an argument line consisting of one UNIX command coupled with that command's associated arguments. Specifically, the usage statement from the manual page reads:

```
priv [ -d ] [ -u user_name ]\
  Unix.command [ args ]
```

Once invoked, Priv will break down the argument line and compare the user given command to a list of usable commands kept in "Priv.conf". An entry in the configuration file will indicate every default user name which may be utilized to execute the stated command. There is one entry for each command which may take one of the follow forms:

1) Command WS Execute path WS #:UN #:UN ...
2) Command WS Execute path WS UN
3) Command WS Execute path WS UN @

- Command: is any desired Unix command.
- WS: represents for White Space (spaces and tabs) which are used as delimiters.
- Execute path: gives the command executable path
- UN: is any valid user name or *.
- *: is the priv program user's name.
- #: denotes an integer argument position.
- @: allows use of the -u Priv option for this command (see below).

There are two possible routes to processing a command in the "Priv.conf" file. First, a single UN within a command entry denotes the only user name which may be employed for that command. The other requests positional entries designate a user name which is associated with the specific number of arguments passed to a Priv executable command. For example, the date command configuration entry may indicate an asterisk for zero arguments (simply reporting the date and time) or root with one argument (which allows the user to change the machine date and time) as follows:

```
date /bin/date 0:* 1:root
```

Priv has a run time option that supersedes the configuration file default. The character "@" may exist at the end of a entry and enables the Priv -u option. The -u option must be followed by a valid user name which will override any default user name configuration in the entry. Thus, the Unix restore facility may be used to reinstate files from backup tapes directly into a user's directory.

The other method of Priv operation involves the use of symbolic links. Using the Unix ln command, a symbolic link may be established with Priv in the name of a configured Unix command. Once forged, the link may be executed right from the shell prompt. The typing of the Unix command will prompt Priv to use the zeroth argument as the executable command name. However, this method requires the root shortcut file name to match with a configured Unix command, and thus, does not allow Priv command line options.

Upon initiation, the desired Unix command is examined for appropriate configuration and the user name is converted into a UID (user identification number). After verification, Priv employs the Unix-C "setuid" statement which alters the UID associated with Priv to the user specified value. Subsequently, the Unix-C "execv" statement shuttles the user specified command arguments to the Unix command program for execution.

Consequently, the nature of Priv allows for extended administrative capabilities without jeopardizing requisite security. This is accomplished by creating several strata of users and administrative personnel with varying degrees of system utility access. In particular, the network users have access only to their respective research programs. Whereas, the "pseudo superusers", which are a chosen subgroup of operators, have the ability to manipulate whole user files, but not the ability to modify existing programs [i.e., destroy but not modify]. Then, there are the "superusers" who can manipulate user program files, modify the Priv configuration file, and access the Unix source code. As a result of this hierarchy, the ability to modify programs remains tightly restricted while the ability to administer the system has expanded, without increasing the associated risk of undetected security breaches.

Priv has been operating in the field for two years without major complaint or modification. Interested parties may obtain Priv via the ARPANET by directing requests to heilman@BRL.

A Centralized Multi-System Problem Tracking System

Ken Harkness, Systems Administrator
seismo!mimsy!aplcen!osiris!ken
Operational and Clinical Systems
The Johns Hopkins Hospital
Baltimore, Md.

The Johns Hopkins Hospital Department of Operational and Clinical Systems has in the past 3 years put together a hospital-wide ethernet network. We have developed a central hospital-wide patient database on one Pyramid 98x computer and have several other applications which access this database for Patient Lookup, Scheduling, and Registration using rpc network transactions (using both XNS and TCP/IP protocols). Systems which retrieve data from our Pyramid include other Pyramids, an IBM mainframe, several PDP-11's running MUMPS, and Microvaxen running Ultrix. Our department is responsible for the operation of 4 Pyramid 98x's, 1 9820, several Sun Workstations and a Sun Server 3/160 as well as maintaining the ethernet. We have one primary systems administrator for the Pyramids and one for the Sun and Network.

On the Pyramid systems, warning signs or problems on systems other than the administrator's "home" system occasionally went unnoticed until a the problem became serious. (An example is a filesystem growing steadily until it fills up.) The system monitoring logs and functions (such as df, and /usr/adm/messages) are certainly available on each system, but it is easy to forget to check these on a day-to-day basis. We created a set of programs to report the output of several system routines to the system administrator each day by mail.

Currently, three items are reported to the system administrator, by each system, each day.

1. A "composite disk report" which checks each disk for bad tracks, using the Pyramid iopformat routine. This is critical, as we periodically develop bad tracks on the disks, and can now correct them before they become completely unrecoverable. (Previous to this reporting system, we had 2 occasions where we lost one track on a disk-wide database and had to recover the entire database from backups.) In addition, this iopformat program is very disk-intensive and would not be reasonable to run during daytime production hours. Instead, it is run at 1am, when system load is very light.
2. A composite report of all new logs to the file /usr/adm/messages.
3. A composite report of the filesystems which are over 90% full or which have changed size dramatically (+/-) over the past day.

Each composite report is a combination of the reports from the individual systems. A "master" program on the "master" computer (the administrator's "home" system) is responsible for combining specific individual system reports (using rcp/rsh to get them) into a composite report and sending it to the system administrator via mail. For each of the reports listed above, one mail message is sent. The "master" program is a shell script which is run via cron each morning. This shell script can be expanded as other reports are needed. The "master" shell script also cleans up the files on the remote system.

Each computer is responsible for generating its own set of reports and leaving them in a "report" directory for the master system to gather. Cron is responsible for running the programs which generate the reports for that system. If a system is down, no report is generated for that day, and the master program gets nothing. Reports are stored in file names with names that are date specific to allow for the days when the "master" system is down. (The next day, there will be a composite of 2 days' reports.)

In addition to the reports, we are also storing some of the relevant data in an Ingres database on the "master" system.

A Cron Facility for Downtime

Ken Harkness, Systems Administrator
seismo!mimsy!aplcen!osiris!ken
Operational and Clinical Systems
The Johns Hopkins Hospital Baltimore, Md.

One of the big disadvantages of cron as a means of running regular jobs is that if the system is down during the time that a job is to be run, execution does not occur. For example, in our environment, we often have to generate daily reports - using crontab entries - during off-system hours. However, once per week the system is taken down for backups (which may take 6 hours).

In order to overcome this problem, we had a "batch_cron" facility developed. Basically, batch_cron is the same as cron - (there is a process which runs continuously while the system is up and checks a crontab file once per minute). After a downtime, batch_cron is started up in /etc/rc just like cron. Batch_cron checks when it last ran and executes all entries which fall into that gap.

Below is the manual page for batch_cron; we cannot distribute any sources.

NAME
batch_cron – clock daemon

ORIGIN
System V

SYNOPSIS
/etc/batch_cron

DESCRIPTION
Batch_cron executes commands at specified dates and times according to the instructions in the file **/usr/spool/batch/crontab**. If the system is down at the scheduled job execution time **batch_cron** is able to recover and execute the missed jobs after system reboot. Because *batch_cron* never exits, it should be executed only once. This is best done by running *batch_cron* from the initialization process through the file **/etc/rc** (see *init*(8)).

The file **crontab** consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify in order:
minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
and day of the week (0-6, with 0=Sunday).

Each of these patterns may contain:
a number in the (respective) range indicated above;
two numbers separated by a minus (indicating an inclusive range);
a list of numbers separated by commas (meaning all of these numbers); or
an asterisk (meaning all legal values).

The sixth field is a string that is executed by the shell at the specified time(s). A **%** in this field is translated into a new-line character. Only the first line (up to a **%** or the end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Batch_cron examines **crontab** once a minute to see if it has changed; if it has, *batch_cron* re-reads it. Thus it takes only a minute for entries to become effective.

FILES
/usr/spool/batch/crontab
/usr/spool/batch/cronlog

SEE ALSO
init(8), sh(1).

DIAGNOSTICS
A history of all actions by *batch_cron* is recorded in **/usr/spool/batch/cronlog**.

BUGS
Batch_cron reads **crontab** only when it has changed, but it reads the in-core version of that table once a minute. A more efficient algorithm could be used. The overhead in running *batch_cron* is about one percent of the CPU, exclusive of any commands executed by *batch_cron*.

Resource Duplication for 100% Uptime

Pennie Hall
ihnp4!tellab5!pennie
Systems Support, Tellabs, Inc.
Lisle, IL

Tellabs is a supplier of analog and digital voice and data communication equipment. At our Research and Development Lab headquarters, we have over 200 engineers developing software on 4 DEC VAX 11/785 machines running UNIX 4.2BSD. The 4 machines are connected via Interlan's 10Mb/s Ethernet. The users' terminals are connected to a particular machine via a data network consisting of our own products. There are 2 system programmers and 2 operators in our support group.

Because the software development is dependent on our machines functioning reliably 24 hours a day, we have set up a mechanism such that if we lose Machine A, we can bring up a mirror image of Machine A that is at most 24 hours old onto Machine B in a matter of minutes. We do this by copying the user and project file systems from Machine A to an empty file system on Machine B via "cron" every night. One of the major requirements of the scheme is lots of disk space and file systems of similar size.

We start by setting up each user and project file system on all machines to be of identical size. By doing this, we are insured that an entire file system will fit on a remote file system. The remote file systems are identical in size or larger than the user and project file systems.

Second, we set up a table that desncribes all user and project file systems and where they will be copied, i.e. from which machine and file system to which machine and file system.

Then, each night via "cron", we run a modified version of "dd" (called "backup") that works over Ethernet. It mounts a remote file system, copies a user or project file system onto a remote file system, runs "fsck", and tracks status information regarding each file system backup.

As a result, if Machine A is down, we move the I/O communications boards to Machine B, mount the copied up file systems and go. Because our data network is set up such that the users login to a particular machine by file system name, the move to Machine B is invisible to the users after minor adjustments to the data network.

Another advantage of this arrangement is that a user can use a command called "getback" to retrieve files deleted unintentionally. This works fine when the user does "getback" the same day he or she deletes the files, since the remote backup will be done again that night. This saves time from having to do restores from tape which could take many hours if the restore is from a level 0 dump set.

There are some potential problems. If the Ethernet is down during the time "backup" is to be run, it doesn't do the copy. Secondly, if "fsck" finds the copied file system to be corrupted, the copied file system is useless. Lastly, it may become necessary to spread out the backups over a large time frame so that the machines don't compete for the Ethernet.

Intelligent Distributed Printing/Plotting

Bruce Spence
ihnp4!hplabs!hpisla!hpltca!bsp
R & D Systems Administration; Hewlett Packard
Colorado Integrated Circuits Division

Colorado Integrated Circuits Division of Hewlett Packard is a designer and manufacturer of custom integrated circuits for use within HP. The R&D Lab provides contract design services in addition to designing IC's. We have seven HP 9000 series 500 minicomputers, each licensed to handle sixteen simultaneous users plus fifteen HP 9000 series 300 multiuser computers. All 22 processors run the HP-UX operating system. The system is fully interconnected via IEEE 802.3 coax LAN and has over 8 GBytes of disc storage. Over sixty users use the system for IC design, text editing and formating, reading notes, electronic communication, etc. The system is centrally administered by two systems administrators.

The system has a variety of hardcopy output devices attached to it at various points including two medium-speed line printers, two LaserJet printers, a thermal printer, an E-size roll-feed drafting plotter, and two B-size magazine sheet-feed plotters, each of which is configured as a spooled device on one or more computers.

The users' printer needs are diverse. Geographical location of printers and plotters is a factor, as is type of output (troff output versus, say, a 200-page raw IC design diagnostic printout). Users homed on the same computer and occupying adjoining offices may have widely divergent printing and plotting needs. A single default printer for all users is not practical.

A big problem is adding a new printer such that all computers in the system have access to it. This requires shutting down the spooler and executing lpadmin, accept, and enable on each of the 22 processors. The lp interface script will not be the same on each computer, as some will access the device remotely by uux, while only one will directly access the device itself.

Our solution uses a hierarchy of processors.

Each non-parent computer has only two types of spooled devices configured: printers actually attached to it (usually none) and the pseudo-device 'rlp' (for 'remote line printer'). The rlp interface script is common across all systems, save for a localized section which describes the lp command string to access printers which are attached to that particular machine. Each non-parent computer has knowledge only of such printers, and usually has direct knowledge of none. The spooled device 'rlp' is in all cases the default spooled device (via lpadmin -d), so that all requests pass through the rlp interface script. Destinations are specified as options to lp, as:

```
lp -odest=laser <file>
```

Other appropriate options may be specified.

If a destination is specified of which the computer has no knowledge, it passes the request up the tree to its parent, using 'uux lp -drlp'. A parent machine at any given level has configured into its rlp interface script all printers at or below its level. If a mid-tree parent cannot process the print request due to lack of knowledge of the printer requested, it passes the request up to its parent. The parent at the root node of the tree has knowledge of all printers in the system. This root parent alone has a specified default printer for requests it cannot service (i.e., an invalid destination was specified.)

If no destination is specified in the lp request, the rlp interface script looks for a file named .dlp_rlp in the requester's home directory. If found, it reads this file for the destination for the print request. If none is found here, the system file /usr/local/etc/.dlp_rlp is read to get the system-local default destination. Thus, each computer has its own default printer which may be over-ridden by any user by specifying his/her own default.

To add a new printer to the system, an entry is added to the localized section of the rlp interface script on the machine to which the new printer is attached, and of the script on the parents above it (at most 3 computers). The lpadmin program needs only be run on the computer to which the printer is attached.

Managing Modems and Serial Ports

Tom Slezak
{sun,ihnp4}!lll-lcc!slezak
Unix Support Team
Lawrence Livermore National Lab

LLNL's Open LabNet is a rapidly growing broadband backbone connecting numerous Ethernet LANs on a square-mile site. Over 50 buildings are currently connected and the completed net will probably encompass at least 200 LANs. The LANs currently are either DECNET or TCP/IP, connected via Applitek gateways to the broadband. An Arpanet gateway is provided. (No protocol translation services are provided to link DECNET and TCP/IP directly.)

The individual LANs contain a variety of machines including every type of DEC box, Suns (models 1-3), Pyramids, Ridges, IBM PCs, Macs (via Kinetics Fast Path), Sequent, Symmetrics, Zilog, etc. An Alliant FX-8 is on order and an SCS-40 is being discussed. Although there is central management for the backbone and gateways, the state of the individual LANs is best characterized as scientific freedom, or complete anarchy depending on your point of view.

One problem we have had is dealing with dial-up lines and printers on the networked systems. Our Vaxen and Pyramids have several hundred users each, with much overlap since not all large packages (S, Vaxima, etc.) are available on each system. Responsibility for obtaining hardcopy output was originally the users', rather than the computer center's.

This policy worked well initially, until the first complement of serial ports (usually 32/machine) began to fill up with modems and hard lines to serial printers. Management balked at adding more serial ports to handle peak periods, and users screamed when access was not available during those times. System administrators of lightly-used machines found that savvy users dialed into their machines and rlogin'ed to the machine of their choice for fastest service. To add to the chaos, we were informed that there were no more phone lines physically available even if we wanted to order them.

Our solution was to build a modem pool using Ethernet terminal servers. A few direct lines were left on each machine (primarily for direct uucp access). Users were enticed to use the terminal servers by putting 2400 baud modems on them and leaving mostly slower ones on the direct lines. As the network grows, users are content to let the terminal server do the connection to the desired machine. The number of people contorted through multiple rlogins throughout the net has decreased since it is trivial to pop back to the terminal server and connect directly. We find that more serial ports are not required and we have been able to provide more remote laser printer lines directly to user locations, solving our printing needs (while leaving printer maintenance chores to the users).

All is not quite so rosy, of course. We have had problems matching the terminal server characteristics (framing, parity, etc) with some users' devices. There are problems with dueling modems, occasional poor service on the net, etc. These problems are very difficult to trace and solve, with so many variables involved. The overall approach is successful enough to make us think in terms of separating out other network functions onto separate servers (we already have a single Arpanet server, a micro-Vax, and are considering having a single Usenet machine. As our load grows, the idea of having a troff/Transcript machine, a Vaxima machine, etc. becomes increasingly attractive.)

The USENIX Association

USENIX, the UNIX and Advanced Computing Systems professional and technical organization, is a not-for-profit membership association made up of systems researchers and developers, systems administrators, programmers, support staff, application developers and educators.

USENIX is dedicated to:

* fostering innovation and communicating research and technological developments,
* sharing ideas and experience, relevant to UNIX, UNIX-related and advanced computing systems
* providing a forum for the exercise of critical thought and airing of technical issues.

Founded in 1975, the Association sponsors two annual technical conferences and frequent symposia and workshops addressing special interest topics, such as C++, distributed systems, Mach, systems administration, and security. USENIX publishes proceedings of its meetings, a bi-monthly newsletter *login:*, and a refereed technical quarterly, *Computing Systems*. The Association also actively participates in and reports on the activities of various ANSI, IEEE and ISO standards efforts.

There are four classes of membership in the Association: student, individual, institutional, and supporting, differentiated primarily by the fees paid and services provided. The supporting members of the Association are:

Digital Equipment Corporation
Frame Technology, Inc.
Matsushita Graphic Communication Systems, Inc.
mt Xinu
Open Software Foundation
Quality Micro Systems
Rational Corporation
Sun Microsystems, Inc.
Sybase, Inc.
UNIX System Laboratories, Inc.
UUNET Technologies, Inc.

For further information about membership or to order publications, contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710-2565
Telephone: 510/528-8649
Email: office@usenix.org
Fax: 510/548-5738